ConvexOptimizationI-Lecture16

Instructor (Stephen Boyd):Let me make an announcement. Actually, the announcement has to do with Homework 8, which we will assign later today. So we are still doing last debugging of it, making sure it works perfectly, or close enough. Anyway, we'll assign Homework 8 later today, it's – this is very sad, it's going to be the last one. I know. We were going to try to put in another one, but that's fine. Of course, I've already had some disappointment expressed about how that's the last one, and how also it's going to be due in 12 days, so instead of a Thursday to Thursday cycle, it's going to go Thursday to Thursday and then after that.

Someone said, "Oh that's disappointing that you have a full 12 days to do it." My response was, "You haven't seen-

Student: We're not complaining.

Instructor (Stephen Boyd):Oh, I've gotten complaints though. And so I said, "You haven't seen Homework 8 yet." Actually, Homework 8 is quite light. But there's some programming in it. It's not unlike what you are doing now. Wait a minute, it's Thursday, y now everyone has looked at – You just typed stuff in and it worked, your Newton problem. Okay? That's not bad. It's not exactly like writing these eight line CVX scripts. I guess, right, where it just works perfectly very quickly.

This one has got some of that. You're actually going to implement your interior point LP solver. And it will actually be, this is quite funny, it will actually be competitive, meaning it will, if you were to pass in like sparse matrices, it would probably work about as well – it would sort of be within an order of magnitude off from the fanciest ones that have 50,000 person hours of development in them. So, anyway, that's what you'll do. Of course, the reason is not that you'll use it. The reason is to completely demystify these things, so that you don't have to imagine that it's some fantastically complicated thing or something like that, because it's actually not. I mean, they can get fancy, but the point it, you can make a basic one really easily. Well, actually you will do that. That's Homework 8.

Okay, so we'll start in on finishing up unconstrained optimization. So actually, I want to summarize the material we covered last time. Actually, let me do that right now, before we jump into the actual lecture. So the summary is something like this. Here's at the highest level, what we did last time. We looked at the gradient method, which is kind of the most obvious thing you would do. You say which way I would go. Someone says, "That's the fastest way down hill. I'll go that direction. I'll adjust my step length." Nothing could be more obvious. The first, sort of, none obvious fact about optimization is this. While the gradient method actually can work quite well, if you're lucky; if your problem is more or less round. It's not too skewed. You know, the sublevel sets don't look like cigars are anything like that.

In that case, it works; it does and can work very well. So you will see people. You may encounter problems like this yourselves, where gradient method works. For extremely large problems, that's good news, because it's just so simple to do. So that can happen. However, the part that's not obvious is this, that generally speaking, those gradient methods don't do very well at all, and in particular, if you give me a problem where it's working well, all I have to do is scale things, and I will slow it, for all practical purposes, to a stop. So that's the summary of the gradient method.

The other important thing to know about the gradient method is it depends on the coordinates chosen. So if you change coordinates, just scale the variables, the gradient method, you don't get a commutative diagram, the gradient method changes. And in particular, by change of coordinates, you can make the gradient method get much faster, or you can make it get much slower, or basically slow to, for all practical purposes, a stop. Of course, the theory says it's going to converge, but that's maybe not relevant. So that's the thing to know about the gradient method.

Newton's method is the next one that we looked at. Now Newton's method is affine-invariant. So, if you change coordinates by an affine transformation, Newton's, you get a commutative diagram. Newton's method works perfectly. So you get a perfect commutative diagram. In other words, if you change coordinates you get the iterates or the changed coordinates version of the iterates had – it works both ways. What this means, for example, it's completely unaffected by scaling. It is affected by scaling, but only at the level of numeric. So really quite horrendous and violent scalings are required to make any difference at all; of course, in infinite precision, it makes no difference at all in Newton's method.

Newton's method works very well. It has a long – the other part of the summary is this, Newton's method, if you look back at the history of optimization, there's even a hold over. A lot of Newton's method was terribly feared, for example in the '60s. And the things that scared people were, for example, solving a set of linear equations, which you must do at each step of a Newton's method. N3 sounded really bad then, and especially even N=100. So again, you have to close your eyes and imagine 1965 or something like that.

You also have to remember that a lot of people teaching optimization, that was when they had their intellectual adolescence. So they were marked and affected by this. So they still will have an aversion to Newton's method; an aversion to Newton's method, because when they were taught optimization, solving 100 linear equations was a big deal. And therefore, they were subjected to weeks and weeks, and months and months, and books and books full of things, all of which were about how to avoid Newton's method.

So times have changed, and we can solve sets of 100 equations how fast, roughly? How would it take to solve for Newton's step with 100 variables? Come on. A thousand? Didn't we cover this last week? Thank you. Yeah, actually milliseconds, but there's only one character in your statement I don't like, the "S" at the end; it's under actually a millisecond. It's measured in microseconds. Just look, 1,000 equations takes a second or

something like that, and so therefore, do your N3, scale it down, we're talking microsecond. You're talking a millisecond or so. So this is not a big deal.

That's not to mention the following; in many practical problems, the linear equations you have to solve have structure. Like we talked about last week, it's banded, it's blocked diagonal. It's arrow. All these things are going to come up in many application problems. So signal processing, communications, finance; this structure will come up. That just makes it even faster to solve these things. So I guess my summary is now, past, maybe in the 1960's it was very important to do everything you could to avoid Newton's method. It's not such a big deal now, especially if you know how to solve linear equations efficiently. So that's the summary of Newton's method. It works really well.

I mention this only because you will bump into people who will be trained classically, and they will tell you that in fact two-thirds of a course is devoted essentially to attempting to get out of using Newton's method. So you don't have to be afraid of that. That's my summary of that.

Okay, so last we time we looked at the classical convergence analysis, and this is a lead in to our next topic. So the classical convergence analysis for Newton's method basically says this, it says – we looked at. The critical thing is the bound on the third derivative. Now we discussed that last time, it's totally obvious that the third derivative is what affects Newton's method. Because if the third derivative is zero, the function is quadratic, and Newton's method nails the answer in one step. If L is small, if the third derivative is very small, it means that basically, a quadratic model calculated one place is actually a pretty good quadratic model calculated at another. And that's exactly when Newton's method is going to work well, because it means that when you calculate a quadratic model, it's going to have a huge range of where it predicts well what the function is. That's when Newton's method is going to work well.

So everyone agrees the key to Newton's method working well is the third derivative being small. That's obvious, okay? Now, the classical analysis does that by eliminating the derivative. Now, this gets us to actually a very – actually to the second shortcoming of the classical analysis or of the – there's a shortcoming of this classical analysis. And let me this, let me put in these terms. Actually, how shall I explain this? Okay, I'll do it this way. You have something like Newton's method, and you have sort of the beautiful math on one side, right? And then you look and you have like C-source on the other. Okay? That's kind of everything we do looks like that. You have the beautiful theory, and then you have C-source.

And what you want, the way the world should be is this, the theory should be beautiful. Like the theory of Newton's method is beautiful. It says basically if you change coordinates, it has no effect on Newton's method, none. Now, when you look in the code or something like that, or in the – I guess you also have the analysis. If you look in the analysis or the code, you actually start seeing things. You look in the code and you'd see ugly things like a stop – you'd look at a stopping criterion, and it would be the norm is less than the sum, you know, EPS. And you have to look at hash defined EPS somewhere

to find it, and it would be some number like 1E-9. And then you'd challenge first and say, "Well, what's that?"

The correct response to that is, "You're not supposed to be looking that closely at my C-source." That's the first comment. But the second comment is, "Well, we tried 1E-10, and sometimes we had problems. And 1E-5 wasn't a high enough accuracy." This is the way – everyone expects the world to be like this right? Where there's some things in the C-source that aren't exactly totally defensible.

So here's the mathematical analysis. Now let's look at this. The question is, the analysis involves norms, but norms change when you change coordinates. So we have a big problem here. The problem is this, in implementation of Newton's method; actually, it's more elegant than the mathematical analysis. Because the implementation is in fact, at lease at first order, affined and variant. It doesn't depend on coordinates. But the analysis depends on the norm. So for example here, if I change coordinates we know Newton's method doesn't change at all. You generate the same number of iterates. If I rescale X or anything like that, if it took 12 steps before, it takes 12 steps now.

But now we go over here and change coordinates. If you change coordinates, everything over here changes. The Lipschitz constant changes, m changes, the complete analysis, and so you ask the person who – I mean, the code works perfectly. It takes 12 steps, change the coordinates it takes 12 steps. Then you ask the person who did the mathematical analysis, in this case Kantorovich, you ask them, "What's your upper bound in the number of steps it took?" You change coordinates, they'd say, well first of all, they'd say something like "It is 5,230 or something like that." That's assuming you even know these numbers, which you don't. But that's the nature of a bound, right? It's typically useless in practice, right?

Then you change coordinates, and then they say, "Now my upper bound is 67, 248 iterations." And you say, "The method itself, the code executes the same if I change coordinates. Why should your bound change?" This is sort of the world is not in balance here, because you have the mathematical analysis is less sophisticated than the code. And it shouldn't be that way; it should be the other way around. It should be the person who wrote the code who has to defend little bits and pieces and things like that, little stuff like that. That's just wrong. What this motivates, if for aesthetic if for no other reason, it motivates the following, we have to have a way to say the third derivative is small that's affine in variant. Because then you're mathematical analysis will actually be as aesthetically pleasing and sophisticated as the method itself.

Student: What are constants L and m?

Instructor (Stephen Boyd):That's a Lipschitz– no L is defined right here. That's the Lipschitz constant on the second derivative; m is the strong curvature constant. It's the minimum on the Hessian. So what we need is a way to say the third derivative is small that's affine in variant. Actually, armed only with that hint, you could come up with several methods. One of them is going to be self-concordant. Because when you first see

it it's going to look very sick. You'd see it, and you'd say, "Who possibly could think of something like that?" But with what I just gave you – well, let's see, how could you say – let's just take a scalar, a function on R. How would you say F prime, prime is small?

Well, you could say this, that's less than L, that's this, okay? That's no good, because I can change coordinates here, and the third derivative – I can make the third derivative, in fact, anything I want. So I can't just do this. So you tend to do something like that. That doesn't work, because they scale. One scale is like the cube, and one scale is like the square when you scale the argument. So that hints, actually, that if you want to put these two on the same scale, you're going to have to put the three-halves power in there. That's the only way to have a way to say the third derivative is small, but which is a way that is invariant under changes of coordinates, right? Okay, so that's your hint. Save that up, because it's coming up.

This leads us to this idea of self-concordance. Not quite sure where the word came from. It was coined by the originators of this. This is from Nesterov and Nemirovsky. I guess it means something like this – we'll see what it means actually. It's the Lipschitz constant of the Hessian, measured in the coordinate system induced by the Hessian, which sounds weird, but it's not really. So here's what it is. It's going to turn out; you're going to get a convergence analysis. This is one of those cases, which is actually kind of why, I don’t know this is – a lot of people feel this, that if you go after just aesthetics, then often very important practical things wills follow. That appears to have often been the case for lots of stuff in physics and in engineering as well.

Actually, it's obvious why it would work, anyway.

So if we go after this idea of redoing the classical analysis, but now not using numbers that depend on the coordinate system chosen. So you want to say minimum curvature, you want to say Lipschitz constant, third derivative, you have to bound it in a way that does not depend on the coordinate system. So here's what's going to happen. It turns out this idea of self-concordance, which we will get to soon, it's absolutely amazing. It turns out it doesn't depend on any unknown constants. So in fact, all the constants will go away. A function is either self-concordant or it's not. Period, it just goes away. There is no constant, and certainly no unknown constant.

You get a bound on a number of steps in Newton's method, it's affined in variant. So finally now, sort of now the mathematical analysis is as sophisticated as the actual method. Now you have something where you change coordinates, and you ask the person producing the bound what their new estimate for the number of steps is, they give the same answer as before. And this is actually the key for establishing that, for example, interior point methods we'll look at next week, they have polynomial time complexity. So that's the general method.

It turns out also that self-concordance; it's not quite clear what role in practice it has. I'll say more about that later, although it clearly has some, that's for sure. So what is self-concordance? Well here it is you say that a function from R to R is self-concordant if the

third derivative is less than – the two is there for a convenience, and can be replaced by any constant you like. It doesn't make any difference. So it says that the third derivative is less than two times the second derivative to the three-halves power. Now when you first see this you think, "Well, who on earth would come up with this as a way to say the third derivative is small? It just makes no sense."

But actually after a while, if I told you you have to say the third derivative is small, and you have to do it in a method that's affined and variant, that doesn't change when you change coordinates, this is actually one of the only ways you could do it. Now, this is much more natural, because they just scale differently. Okay, so that's what it means to be self-concordant. All right, then you say that a function of RN is self-concordant if it's self-concordant when it's restricted to any line. So that's what this means.

So, let's look at some examples. Linear and quadratic, that's easy because the third derivative, everyone agrees that liner and quadratic functions actually are very well suited to Newton's method; actually, linear less, because Newton's method fails instantly. But quadratic, everyone agrees that's perfect, because the third derivative is zero. Negative logarithm, it turns out this satisfied this. So if you just work out – well, it's not exactly hard to do, you take the third derivative, and whatever. I think it's actually right on the boundary or something like that. You get maybe equality or something close to equality here.

Anyway, whatever it is, it's extremely simple to verify. The negative logarithm is self-concordant. It turns out there's lots and lots of examples. One example would be something like entropy, I'm sorry, not entropy. Entropy alone is not self-concordant, but if you add an extra log term it is. Then you have affining variants, this is the whole point. If you change coordinates; in the scalar case, you just scale and shift. The shift doesn't make any difference, but the scale by A does the following: the third derivative scales by A3, the second derivative scales by A2, the three-halves here and sorry the three-halves power here, basically makes the A drop away. So if F is self-concordant, so is F of AY+B.

So now you have various – you have a self-concordant calculus. And so here are some basic properties, 1.) It's preserved by scaling, but only if the scaling number is bigger than one, so scaling a function by a constant bigger than one preserves self-concordance. 2.) It's preserved under affined composition. And there's a bunch of other stuff we're not going to go into too deeply. But for example, there would be a generic result says that if the third derivative is less than three times the second derivative divided by X, then this function here is going to be self-concordant. We're not going to go into details. There's more on it in the book and so on.

Let's look at some examples. The shocking part is that many functions that come up that you will encounter are self-concordant. By the way, not all, for example entropy, so entropy is not – one over X is not, for example. And all sorts of other things, but some are. Here are some examples; log barrier is self-concordant. Log determinant of a positive definite matrix – by they way, these are just not obvious things, like remotely, to show.

First of all, you don't even want to think about the third derivative of this function. I don't recommend it. I'll tell you why, the third derivative of log det. A, you barely want to think about the gradient. The gradient is a linear function on symmetric matrices. That's okay, because that has the form of another matrix with an inner product.

The second derivative is a pain. I don't recommend thinking about it. The second derivative of this function here, right, is a bilinear form on symmetric matrices. It has four arguments, so unless you're used to dealing with tensors and things like that, and can handle them safely. I guess if you come from mechanics of physics or something like that. I don't recommend it.

Third derivative, make me laugh. The third derivative here has six indices. Everyone agree? So unless you are specially trained and certified to think about things with six indices, I recommend not doing it. So that's my recommendation. Nevertheless I've a hard time just thinking about what the third derivative is; nevertheless the third derivative is less than twice times the second derivative to the three-halves power. And actually using the calculus here, you can show this. Other examples, this is the so-called barrier for the second order cone. There's plenty. We'll get to that.

Okay, so why do you care about self-concordance calculus? Actually, I'll come back to that question several times, because the answer is a bit subtle, and it's important to understand, so convergence analysis for self-concordant functions. This is just a different convergence analysis for Newton's method. So by the way, if your interest is entirely practical, then none of this is of any interest to you. In the following sense, we already know Newton's method A.) Works really unbelievably well in practice. That's point A and B.) We know that we have a proof of convergence. It's true the proof of convergence involves constants that you don't know and will not know No. 1. And No. 2, even if you did know those constants and plug them in, you get an upper bound on the number of steps that's ridiculously high, and therefore useless.

Now if you're practically oriented, and say, "I didn't care anyway, what I care about is that I can solve these problems with 100,000 variables in 15 steps vary reliably." Which generally speaking, is true; it depends on the function, of course. So if that's your interest then the only thing you want to know about this material is how would you show? How would you prove? How would you actually get a real number that says "That's the number of steps it takes to solve a linear program." Anyway, this is what you would use.

So here's what it says, everything depends on lambda. That's our old friend the Newton decrement here, and if it's bigger than the number ada, and ada – actually you can even put in – I mean, you can get very deeply into this, and ada would be 0.09 or something. You can get very deeply into this if you want to optimize the analysis. And it says the following; it says that the function value decreases by at least a number gamma if the Newton decrement is bigger than ada. Okay, and then it says once it's less than ada, it says twice the Newton decrement is less than twice the Newton decrement squared. Okay?

Now putting these two together you get the following, it says here's the number of Newton iterations. It says it's less than the initial function value minus the optimal value divided by gamma plug log, log one over epsilon. Okay this we read as five or six, I don't care, but that's five or six. That's it. Okay, now here's the most amazing thing. These numbers here depend on alpha and beta. They're not hard to work out. If you use typical numbers of alpha and beta, and epsilon is 10-10, it seems like a quite reasonable final stopping criteria or something like that. Then the numbers work out to 375 + 6. That's 375 times the difference in the function plus six.

Now by the way, with some work you can actually analyze all this stuff. And instead of doing very sloppy bounds, you can actually come up with a number that's around 11; f(X0 - P*+6), six is our way of writing this, that's our special way of writing that.

Student: This is never [inaudible]? Instructor:

An upper bound, that's the number of bound when the number of steps. All right, so you get this thing here. Now here's the wild part, and this is really cool right now. Because it basically says, look here, several things are absent from this. If you look carefully here, you don't see n, you don't even see the dimension; you don't even see – n is gone, m is gone, L is gone. In fact, you can scan this sheet for a while, and it only depends on two things, that's alpha and beta. These are parameters in the line search. You know them because you choose them. So this is really one of those exceedingly rare things. This is a complexity analysis, which actually passes the first step in actually being useful beyond conceptually useful.

In the sense that most complexity analysis looks like this, I walk up to you and I say, "Listen, if, " and I give you a long list of hypotheses, if this holds then, and I give you a long list of conclusions. Then usually the hypotheses are unverifiable, and the conclusions are uninteresting. And then when you point that out to the person telling you that, then they say, "Oh, you've missed the whole point. You've totally missed it. This is for conceptual. It's conceptually useful." And then you go, "Oh, okay. Thank you." And they go, "Can't you sleep better now that you know that if some things that are unverifiable were to happen, then some things that are interesting would happen?" And you go, "Yes, thank you very much. I will sleep much better. Thank you."

I'm not making fun to it. It's better than the opposite where you just kind of code stuff up and run it. It is nice to know that if you are confronted by a gang of angry theorists in a back alley somewhere, and there's no way out. You can actually justify it. So that's good to know.

But what's amazing here is these are just numbers. It just says basically if you are minimizing log data, if you're minimizing this log barrier – oh sorry, pardon me, aren't you doing that right now? So aren't you all happy, now, to know that in fact that is true? That for your codes, the number Newton steps, I hope, in no case did it exceed this.

Student: For the initial question value minus the optimal value, isn't that totally dependent on your choice of coordinates?

Instructor (Stephen Boyd):No, you can multiply F, that's one thing. The choice of coordinates scales X inside. If you multiply by something X, that's a change of coordinates. If you multiply F, then yes it does. It changes everything. However, I'm allowed to scale F up, that just makes by bound worse. That's why I can't scale it down; that's why this is the case. Otherwise, I would say really? It only depends on the difference in my starting and final value? No problem, I'll multiply my function my 1E-10, and minimize that. Then this goes down to 1E-10 and its six steps. But it doesn't work that way, because you're not allowed to. You can scale it up, good question though.

It's really quite beautiful, and it's good. Order is restored in the world, because now your complexity analysis is at least aesthetically as elegant as the method itself. So that's good. Yeah?

Student: Does it depend on alpha and beta?

Instructor (Stephen Boyd):You can check it. Actually, it's relevant how it depends on alpha and beta. It doesn't matter how this number changes. Student:

Wouldn't it change how fast that thing grows?

Instructor (Stephen Boyd):Yeah, it affects this. It does affect that. All right, so you can look at – you can actually try this out. Indeed, you can if you want. In fact, what's going to happen since you all are running Newton methods today, yesterday, last night, later on tonight, you'll be running Newton methods. I don't think anyone here will find an instance where it takes, I don't know, more than 10 or 15 steps or something. Has anyone found any instance where it took more than that? Actually, how many people have gotten their Newton thing up and running? Okay, what's the typical number of steps? Anyone scale things up big? Did anyone try a several thousand variable problem? What's that? Did you?

Student:[Inaudible]

Instructor (Stephen Boyd):Okay, yeah but you know how you can do that, right? You just make a sparse, right? You make a sparse, I guarantee your code, exactly as it is, will go up to a 100,000 variables. Just make it sparse, and don't make any mistake that promotes a spare matrix into a full one, because if it's 100,000 X 100,000 it will just stop right there. It will attempt to allocate memory and fail.

So it's an interesting thing. You can play with it and all that. What's cool about this is first of all, that just such a bound exists. Because this is very, very unusual; it's very unusual to have a bound like this. Even though it's not that relevant, even at 11, it's not such a great bound. Here's what's interesting. Here's an example, I think these are just log barriers, so this is exactly what you're doing. And here's ones where you just generate a

random instance, solve it. By the way, after you solve it, you know P*, so you can plot P*, and you can jot down the number of Newton steps required. I think these are various problems in various dimensions. It would look no different; actually, it gets more favorable the larger the problem, but it doesn't matter.

And you can see various things here. This bound wouldn't even fit on this, it would go from six, it would go straight off. It's almost veridical, the bound. Good news, all these points are below the bound, that's good. The interesting thins is, at least here, you can see something that looks like that. It looks like the constant in front here actually might be on the order of one or two or something like that. That's just empirical. No one can prove that. The best bounds – with all sorts of work you whittle this 375 down to about 11. Probably if you worked harder you could make it smaller. But the point is it would appear that if you replaced 375 with 2, you get a pretty good empirical estimate of about – by the way, what are these?

What's that? Well, it's a problem with a pretty healthy f(X0-P*), which is to say a pretty crappy starting point, but Newton's method got it pretty quick. So what do you think, what is that? What does everybody call that?

Student: Luck.

Instructor (Stephen Boyd):Yeah luck. I think luck is the right answer. So basically, remember there are actually points way, way out there, way far away from optimum, here just as fate would have it, the Newton direction points you directly to the solution, right? There are such points. I'm not saying there is a lot of them, but there are such points. But the point is this is just luck. You started far away, and for some reason the third Newton step just happened to be in a really, really good direction. You made much better progress than you should have, and there you landed. So these are luck.

Actually, what's very interesting about this is because it says that when we do complexity analysis, which we will do next week, it says actually you can give the same lecture, you can talk and do analysis, and actually it's interesting, you can talk to two groups of people at the same time. You can talk to – your complexity theorists can sit on one side of the room, and then – but what we'll do is we'll define this to be a constant C. On the left side of the room we'll have the complexity theorists. They'll say, "C is 375." And we'll have another one called D, and we'll say D is log, log one over epsilon. Okay? They'll sit over there, and we'll argue using C x (F-P0 +D).

Other people will sit on the right, and they'll say, "Look, I couldn't care less. I know it works, obviously it works. I'm totally convinced. I've solved 1,000 convex operational problems this quarter. None of that would have worked if this didn't work. I'm convinced. I'm not interested in the actual complexity. What I'm interested in is things like this, how should I choose the parameters to make the algorithm run fast. And I don't want some bound like 5,000 iterations. I want to know, how can I make it 22, as opposed to 66?" That's the other extreme. They have no interest in that.

You can talk to them, because you can say, "Oh, no problem, we'll make this one and we'll make that six." And they'll be satisfied, too. Then we'll optimize the coefficients and things like that. You then actually optimize algorithm parameters. We'll do this next week. We'll find out what the optimum are for the complexity theorists and for the people who are interested in the practical methods. And they'll be different.

All right, let's talk about implementation, which is where linear algebra from last week comes in. So the main effort in each iteration of Newton's method – it depends of course on evaluation the gradient and Hessian and all that, but assuming that in most cases that's modest compared to actually solving. But that doesn't have to be the case. In any case, that's modest compared with solving for the Newton direction. You have to solve a positive, definite linear equation. That's it, so you solve linear equations here.

How do you do that? Simple answer is if H has no structure, or whatever, Cholesky. So you just do Cholesky here. You do a Cholesky factorization. You do a backward and then a forward substitution, that's this, and so on. And then as you are doing various calculations for example, you get lambda is the norm of L inverse G. So, when you actually do this, if you did a real implementation, you'd actually find out that some of the things you're computing anyway are useful, like you'd get lambda here. And the cost is one-third N3 flops.

Now if H has structure, then of course, you can solve this much faster. For example, if H is circulant. If H is banded, if it is banded arrow, any of these things you can exploit and solve this way, way faster; way faster than N3, which, by the way, already is not too slow. S to give an example where you would have a dense Newton system with structure, well actually, I guess on the current homework you're looking at a problem. Its quadratic, but in fact solving that is one step of a Newton method. And that's an example, you'd see, the structure would come up. In a linear algebra problem, the Hessian structure reflects sort of which variables couple nonlinearly to which others. And actually after a while you'll get used to looking at a problem, and just say, "I know the structure."

So if you look at signal processing, you can look at other problems, and you'll just know immediately what can be done. Another one is sparse. What does it mean if the Hessian is sparse, by the way, for a function? What would that mean? I have a function, and the Hessian is sparse, what does it mean? It has a meaning, a very specific meaning. What does it mean?

Student:[inaudible]

Instructor (Stephen Boyd):Exactly. Right, it's almost – it means that each variable couple nonlinearly only into a handful of others. That's roughly what it means. And by the way, if they decouple nonlinearly it means it's separable. It means it adds up. That's what it means. When would you get things like that? Well, that's kind of a vague question. What would be some examples of when you'd have a sparse Hessian? What about when you saw something that looked like this? I'll just make it up X of XI – (XI +1) there. Okay, there's a highly nonlinear function of a bunch of – on our end right?

That's highly unlinear. Who knows why you'd every want to use this, but let's imagine, there it is.

It's some weird thing; I guess this is a smoothness measure or something. Who knows why? It's bad, XI can be above, this could be positive, but otherwise XI – anyway, it doesn't matter. I won't even attempt to make a practical problem where this comes up. But here's what I want to know more than anything. For this thing, what does the Hessian look like? Tri-diagonal, exactly, so if you apply Newton's method to something like this, how fast is it going to be?

Student: An nth.

Instructor (Stephen Boyd):N2? It's going to be N, because tri-diagonally you can solve super fast. So if you were in something like this, and you go, "No, no, I work on medical imaging. Our problems are really big. So you can take your Newton methods, and put them in the garbage can as you come in. Because other people in other fields, they work on smaller problems. We work on really big ones." Okay, non issue. You could take 30 million variables, you could run full Newton's method, provided this is not medical imaging. This would be some signal processing type thing. But provided you understand, you recognize that the Hessian is tridiagonal here. Okay.

Here's an example of a dense one. This, hopefully anybody would get. Although you'd be shocked at how many people would not, including especially large numbers of the people who would tell you all about the complexity theory of Newton's method on and on. Many of them would have absolutely no idea you could solve a problem like this with Newton's method very fast. Nor would they have any idea how to do it, I might add. I'm saying that naturally, because hopefully their students will watch this and go tell their advisors, this is in other places.

All right, let's look at an example where you couple the material from last week with this one. Here's an example. This could easily come up. So here, you want to minimize the function that looks like this. It's a separable sum; it's a sum of a bunch of scalar functions of the variables. This could be a regularization thing. Who knows what this is? Some sick thing here right? It might involve some negative log likelihood function. I don't know. Then plus a non linear function of an affined function of the variable, so that's your function. How do you minimize that? Well, we'll assume here that this function is small; the size of this thing is small. And in this case, if you take the Hessian of this, the Hessian of a separable function is diagonal. So this thing, you get diagonal.

By the way, that should light up all of the senses – you should have a section in your brain or will, I hope, have a section in your brain devoted to equation solving complexity. It's sort of like – it will be near the visual cortex or something. It should light up big time when you see a separable function, because separable function should mean for you diagonal Hessian. Diagonal should be right at the top of the list on structures that should be exploited, obviously. By the way, anybody can do diagonal. So that's easy enough, right. If someone said, "How would you minimize that?" In fact, someone answer me,

how would you minimize this? The sum, the functions, you minimize each one separately right?

So interestingly, what would be the computational complexity of that? Roughly? Order in? Right, this makes it interesting, because now the Hessian has this form. It's diagonal, plus A transpose H zero A. H zero is the size of the height of A. So for example, if this is 10,000 variables, and that's 1,000 here, let's say, or a 100. Let's make it 100. If this 100 and that's 10,000 that's diagonal plus rank 100; its 10,000 diagonal plus rank 100. This should be lighting up that center in your brain devoted to smart linear algebra, to being able to solve that. That should set off everything. That's diagonal plus low rank or something like that.

How do you actually solve this? It's pretty straight forward, you would factor H0, do a Cholesky factorization on H, that's the small one. So if this 100 x 100 that goes down in microseconds, maybe a millisecond, but fast. You do this, then you – this is just the elimination done with Cholesky factors. You write it as D delta X plus this, where you introduce a new variable. The new variable is going to be H zero A. So you write it this way. You factor this. You can solve these. You can eliminate delta X from the first one and so on, and you end up forming this matrix. But that's the small dimension, that's 100 x 100. Now you probably haven't done enough of these to know immediately what the computational complexity is, but it turns out, actually solving this 100 x 100 system is zero.

The dominant cost is actually doing this multiplication right here. That dominates. It's a matrix multiplies. So it 2P2N, but in particular, if this was size 100 and that was 10,000 this would go down like really, really fast. Okay? And it's interesting, this is a great case to contrast, if you go back and imagine like the Apollo 1 or something like that, go way back and imagine whole books attempting to avoid Newton's system, because a Newton system with 50 x 50 was like a big deal. And you just did it by a 10,000 x 10,000 dense system. Now you never form the matrix H here, never. If you did, it would be all over. Forming that matrix, first of all you couldn't store it; 10,000 x 10,000 you could. Make it 100,000 x 100,000, now you can't store it. It would take you too long to compute it and things like that.

So this is a perfect example where – there's not that many people by the way who would know this. Even though what we are talking about now is only putting together six or seven things, which is not particularly complicated. You can solve this problem really, really effectively if you know what you're doing. That's this one here. That's just one example; there are tons and tons of others.

That finishes up our tour of Newton's method.

Student: How would you determine the functions, would those be given to you?
Instructor:

Oh, these are given to you. Yeah, these functions are given to you. By the way, you have to know F has this form. If someone says, "Please write a Newton method, but the only thing I'm going to give you is the following, I'm going to give you a pointer to a function that will return the value, the gradient, and the Hessian of F." If someone does that, you're screwed. This won't scale to more than 1,000, and it will be super slow. So the point is, if someone gave you this matrix. That's end of story, it's all over.

Once they've given you this matrix written this way, you can't do anything. There's nothing you can do. The key is to know that it is has this form. So if someone else is creating these matrices, they have to pass you a data structure that gives you D, A, H zero.

Student:[Inaudible]

Instructor (Stephen Boyd):No, just to calculate H that's all you need to know. That's all you need to know. Any more questions about this?

Student: Quick question. If [inaudible] what if the second derivative were negative?

Instructor (Stephen Boyd):I got asked that question by someone very famous once. The functions are convex. That's a very important point. So self-concordance only applies to conve – actually, it's interesting. If you look back at the inequality, you'll realize a function could only be if it's self-concordant, that subsumes convexity. Let's see if I can find that inequality. I'll find it somewhere. Oh, here it is. So one way to say it, well no I'm sorry, the right way to look at this is the following – I guess we should say, I guess to make this precise, we'd say convex there. That's a good point.

So what you would say is something like this, the three-halves power doesn't make any sense at all if it's negative, unless you're in a complex analysis class. So the three-halves power, for this even to make sense this part here, you have to have F prime, prime positive, non-negative, which means it's convex. Okay.

Student:[Inaudible]

Instructor (Stephen Boyd):That's a great question. The question is can you detect structure plus low rank or something. It's actually a huge field, there's a lot written on it. And let me tell you what I know about it, which is a strict subset of what is known about it. And I'll tell you this, there's some obvious cases where you should be able to do it. For example, if you can detect if a matrix is a multiple of an identity, plus a low rank matrix. Because that you would do by eigenvalue value decomposition. That one you can actually detect.

Although it's true that eigenvalue decomposition would cost you n3, right, which would kind of ruin the whole thing, unless you're going to use that low ranked structure and you're going to amortize it across many, many uses. There is actually ways to detect if a matrix is a multiple identity plus a low rank. Diagonal plus a low rank is actually a

famous kind of semi-open problem. I know that there is a lot known about it. Some nod and things like that. It comes up in statistics actually. It's a big deal. I forget what it is. It corresponds to determining factors where you don't know, it's a factor model, and you don't know the individual variances.

So for example, to write this where that's a covariance matrix, this is a very simple interpretation, it basically says that there are factors underlying the vector observations you're seeing, a small number of factors. For example, let's do this in finance; that would be a perfect example. In fact, that's where models of this are extremely common. So one factor would be the whole market, another factor would be some sector or something like that. But the fact is there might be like five of them. The diagonal here, if this is a returns covariance matrix, the diagonals are interpreted as the firm, specific variances. And they are independent. So there would be a lot of interest in writing models like this.

I guess someone in the class actually pointed out the following. If this is rank one, you can actually determine if a matrix has diagonal plus rank one structure. That you can check. But in general, like the low rank structure, you can't check with a diagonal here. Other things, it's even more complicated, like how would you – I actually think that, since you asked, I'll actually say a little bit about it. I think the situation here is like disciplined convex programming. So when you first do convex optimization, what you want to do is to write a modeling language where you can write down anything you want, and then some program will analyze it, come back, and say, "Oh it's your lucky day. It's convex."

That's kind of the most natural thing. You write what you want to do, go to lunch maybe because it might take a long time, then it comes back and says, "Hey you're in luck, you wrote a convex problem." If you think about that, you can actually implement things like that. They actually do kind of work and stuff like that, but it's completely the wrong way to do it. Because when you do that, when you're typing in your problem, it's like a monkey at a typewriter typing in something hoping it's convex. If it's convex, you don't know why. You don't know which signs in your problem you can change with no problem whatsoever, and others will completely screw it up. You know what I'm saying? So it's better to do discipline convex programming like you've been doing. I think, because it means you know what you're doing.

I think maybe someone should make something that's disciplined linear algebra, where the goal is not to discover some structure that you or one of your fellow researchers obscured my multiplying this out. This doesn't apply to raw data, by the way, of course. There it's not that somebody obscured the sample returns, this is a model. But in the case of working with stuff, you would actually want something like disciplined linear algebra, where you have an interesting data structure. You would store things as diagonal plus low rank, and that would actually be trapped and taken care of. That's actually what you'd really want, to tell you the truth, something that works that way. Not something that kind of discovers structure.

The same way with convex programming, I think it's better to be given a set of atoms, a set of rules that come straight from the math, and you just follow those and everything's cool. That's my thoughts on it. Was there any other question on this?

Okay, we'll turn to equality constrained minimization. So this is going to turn out to be not much different from the unconstrained case. If we want to minimize f(x) subject to AX = B. We'll assume A is fat, and it's full rank, I mean, in particular AX = B is solvable. And we'll also assume that the problem is bounded below, and f is convex and continuous is twice differentiatable, a continuous second derivative. And the optimality conditions are simple. It's if and only if there exists a new, such that the grade plus A transpose new equals zero. So that's the classic Lagrange's multiplying rule. However, in this case it's if and only if, right? There are no constrained qualifications of any kind here. You don't even have to apologize. These are the necessary conditions.

Another way to think of it is this, is you really want to solve a set of N + P equations in the variables X* and new*, which just coincidentally happen to have dimension N + P, right? Because this is N equations and that is P equations. You can say that the same way, when you do unconstrained minimization of f, you can say, well really my job is to solve the N, nonlinear equations and N variables; nonlinear, by the way, unless f is quadratic, in which case this is linear. But in general nonlinear equations, that's what you're doing; N equations and unknowns. That's what Newton's method is really doing, for example.

Here, it's not much different. In fact, you want to solve N + P equations and N + P unknowns, but there's actually some structure here. For example, the P, the added P equations are linear. These are the nonlinear ones. And indeed, they are linear in the new. So it's not a completely general set of nonlinear equations. Okay, let's look at the most important case, quadratic. If you minimize a quadratic subject to an equality constraint, this is a positive semi-definite; we'll get to that though in a minute. Then you write this out, and you say the gradient, you just write out these conditions. The gradient here is did I put a half in? Yeah I did, so the gradient is PX + A transpose new. It's a set of linear equations, which you write this way.

The top – by the way these have names. The bottom part is actually primal feasibility. That's exactly what it is. That's what this bottom part is. It says AX* = B, well fine. The top one is actually dual feasibility, because to say this says that new is dually feasible. That's exactly what this means. This is dual feasibility. So this is the condition – by the way some people call, if you take PX* + A transpose new here, and say + Q or this thing – some people referred, there this thing, as the dual residual, and they would call that RD. And they would call AX – B or B – AX, it doesn't matter, they would call that the primal residual RP. So the optimality condition is that RP and RD should both be zero, the residuals.

Back to the quadratic case, when you write this out, you get a set of linear equations like this. Now, that's not a big deal. The solution is this matrix, inverse times that, this matrix has a name. It's called the KKT matrix associated. This is the Karush-Kuhn-Tucker. That's what this is, very famous. In fact, people just call this KKT matrix. It comes up in

lots of fields. By the way, it doesn't just come up in optimization. It comes up in mechanics and all sorts of other areas this comes up. And it's an interesting matrix. It looks like this. Its bottom is a zero. Here, it's got an A and an A transpose here, and it has a positive, we'll assume positive definite and positive semi-definite 1 X 1 block. So that's the so called KKT matrix.

It's non singular if and only if the following is true. On the null space of A, P is positive definite. That's the condition here. That's exactly the condition. And that's the same completely equivalent to P + A transpose A is positive definite, period. And this makes perfect sense. It says that here A transpose A is positive definite, of course. This thing can have – the null space of this is exactly the null space of A. So what you need is you need P to step in and be positive on the null space of A. I'm just reading this equation for you. That's why these are equivalent. So this, the so called KKT for minimizing a convex quadratic with equality constraints; it's a set of linear equations. By linear equations already with structure, it's got blocked structure. It's got this weird 2 x 2 block structure.

Okay, now you can eliminate equality constraints; that's straight forward, and it's often, sometimes, the right thing to do. That works like this. You write the solution set of a set of linear equalities, this is in so called constraint form. People call that constraint for, because it's a set of X that satisfies some equations. You can also write it in so called free parameter form. If you write it in free parameter form, you write it this way, it's X hat or sometimes XP, well P collides with that one. But anyway, it means particular; it's a particular solution of AX = B, plus an element of the null space. Obviously this is the case, set of X's to the AX equals B, is equal to X particular plus the null space of A. SO that's the condition.

Now here to write this out more explicitly, you just calculate a matrix f, whose columns span the null space of A, then this would give you this free parameter representation. Now what you do is no problem. You change variables, and your equality constraint problem becomes this unconstrained problem. Because now you have a free parameter representation of any X that satisfied AX = B. That's this thing. And you can work out various things.

It turns out, if you really wanted to solve this, if you had to write a code, for example that solves this, or something like that, or solves the original problem here, that solves this thing, and someone tell you that your code must return new* as well as X*. And you say, "Why?" And they say, "Because we don't trust you." Because whoever calls this, this is super highly critical, and whoever calls you is going to add something that checks that primal and dual residuals, dual and primal residuals just to check you. So you must return a certificate, and that's the way we do things here. Internally you solve it by elimination. You can actually still return the certificate, because the certificate is just this. You can work that out. That's exactly what – you can check that in fact these things work.

Although you solve it by elimination, you can actually return the certificate. So let's look at an example here. The optimal allocation with resource restraint looks like this. So you minimize a sum. Basically, I have a set of resources, that's our total amount of resource,

that's B, and I have to divide that up among N users of the resource. I guess these can be negative as well, but I don't really care. And I want to minimize some cost of this thing. But the costs are separate. So I want to minimize f1 of X, and each person has a different cost function. Some care about getting a lot. Some care less or whatever. And I run an allocation that sort of minimizes the total irritation or cost across the population while respecting my resource constraint. So that's this problem here.

What you can do is very, very simple. What you do is this, imagine X1 though X (N-1) is free variables, once you've fixed these, then XN is simply B – X1 like that. By the way, this is the same as choosing f to be this matrix here, and X hat is B (EN) here. That's the particular solution. The particular solutions give all the resources to the nth user. It is a solution – sorry, it is a feasible point. The reduced problem looks like this, which we could then minimize now. By the way, when you see this, and if I asked you, just going back to what we were just talking about not very long ago, if I asked you to use Newton's method to minimize this, what would you say? Does it look familiar? Tell me about the Hessian of this, diagonal. Can you tell me about the Hessian of that? I know what it is; you might not be able to do it in your head. What is it?

Student: Rank one.

Instructor (Stephen Boyd):It's rank one, it's an outer product. So it tends that if you wanted to solve this problem by Newton's method, if you didn't know what you were doing, that would be the most common case. Not of course for you, but for people who haven't taken this class or an equivalent one. You would say, "Well, that's N3." But you now know this resource allocation problem can be solved in order N. Because it's going to take 165 Newton steps, like max, and each Newton step can be solved in order N. Why? Because the Hessian, though dense, is diagonal plus rank one, I think I'm repeating myself. I am repeating myself. But that's okay, because these are really important things. That means actually you can do this resource allocation with a million, a million users, it's absolutely no problem. You can do it in Mat Lab, if you know what you are doing.

Okay, so this is elimination. So there's another method, it turns out it's only nominally another method, because it turns out to be equivalent, and that's to directly extend the idea of Newton's method to problems with equality constraints. That looks like this. What you do is you look at the Newton step, and there's lot of ways to explain what it is, it's an extension of it. But anyway, you write down the KKT conditions where you put in the local, the Hessian. So your write down, that's the KKT matrix. That's a step in your X and W is a dual variable, and you write this. This, by the way if there were no – let's see, can I do it? Almost, I did it, there we go. You can see that without equality constraints it reduces to the Newton step, because it's simple the Hessian times the gradient, Hessian times the Newton is equal to negative gradient. That's Newton.

So it reduces to that, and here are some interpretations. One is this. It's completely natural. It says, I'm solving this problem, minimize f(x) subject to AX = B, and it goes like this. The first thing you do is this; you have a current X, which is a guess. And it's feasible, so it satisfies AX = B. And I want to know what should I do? Well what I'll do is

I'll develop f into a second order tailor series at my current point, and I will get this thing. Now that right there is a quadratic function of V. I'm thinking of a tentative step, V. Okay? So I would get this. Instead of minimizing f which I don't know, I'll minimize my quadratic model, but subject to AX = B. Well, AX = B, so AV has to be 0, so V has to be in the null space. That makes perfect sense.

If you are at a point, which is a solution of AX = B, and you are considering directions to go in where you will still satisfy AX = B, you have to move into null space. There is nothing else you can do. Therefore, if you work out what this is, it will say AV = 0, you must move in the null space. And the second one up here, this will be a quadratic, and if you actually work out the conditions for that quadratic, it's exactly this. Notice that the second term here says A delta X equals 0. It says that you will only at – it will always generate a direction, which is in the null space, which means it won't mess up your equality constraints. If they're already satisfied, they will remain satisfied actually independent of the step length you take, because you're moving in the null space.

Another way to do it, to interpret the Newton step is this, is you look at the optimality conditions, which is this here, and you say, look I'm going to take a tentative step, which looks like this. This is my tentative step. In fact, if I could find V and it satisfied this, I would be done, because X + V would be optimal. The problem is that this thing here is nonlinear. So what I'll do is I'll have an affine approximation of this, and I'm going to call that the gradient at X plus the Hessian at X times V. I will replace this expression with that. If I plus this in here to these, I get the Newton step again. Now that's the Newton step for equality constraints.

Now in this case, there's a Newton decrement as well. And there are many formulas for the Newton decrement, and I have to warn you, many but not all of them actually are valid in the case of unconstrained. Some of them are invalid in the case of the constraint. So correct formulas are these, that's one; that's another one, another one. Another thing that's valid, it has the exact same interpretation; one-half lambda squared is your predicted decrease in f. By the way, this tells you that if lambda is small, then your decrease in f is going to be small. It predicts a small decrease. However, this formula is completely wrong in this case, it's just not right. So you have to watch out with these.

And then let me just say what Newton's method is for equality constraints. It's very, very simple. You start with a point that satisfies AX = B, and it's in the domain of f, and you compute to do the step in the Newton decrement. You quit if the Newton decrement is small enough, otherwise you do a line search. Backtracking is fine, and then you update. It's a feasible descent method. It's affined in variant, so you can change coordinates. It makes absolutely no difference. You get a commutative diagram. In particular, scaling has a second order effect on the method. And in face, if it's all done in infinite precision arithmetic, which it never would be, but imagining it were, it would have no effect whatsoever on the method, scaling.

Okay, it turns out we don't need to do any theory for this, because it turns out that Newton's method this way is identical to Newton's method with elimination. You get

exactly the same iterates, therefore everything we know, self-concordance, classical analysis, a practical performance, such as you've found out a little bit about so far, it's all the same. But you might protest, and you might say, now wait a minute, when you do elimination there's lots of choices for f, f is any matrix whose columns span the null space, or our basis for the null space of A. So there's lots of ways to do elimination, surely all those methods can't give you the same iterates. The answer is they do, of course they do in Newton's method, because that amounts to a change of coordinates, and it's completely insensitive to that. So they all give exactly the same.

So you don't need any separate analysis or anything like that, so that's Newton's method. And you will get to apply Newton's method of inequality constraints on your last homework. It's not that hard. By the way, I'm curious how many people, just for fun, actually poked around on the web to look for source code? There's no dishonor in it, you should do this. For their current homework, did everyone write it from scratch, its fine? I told you last time, it's not like it's a secret or anything. Three clicks and you can go to Google and if you type the right phrase, you don't even have to type the right phrase, you'll find source for it. You should only do that if you get stuck, or if you want to look at our implementation. It's all on the web; all of the source code for the entire book is on the web. So you can find our code, then you can see what you like better, yours or ours, or something like that.

I'm basing this only if you get stuck. There's no point in banging your head or something like that on these things; same for this stuff. I want to mention something about Newton method for infeasible points. It's a very interesting method, and it works like this. It says suppose – and by the way, this is now a so called, it's going to lead to an infeasible method. You have the idea of a Newton step even when you are not in the feasible set. That's weird, and let me even just explain, draw a picture, and I guess we'll cover this next time, but let me just draw a picture of that. Here's your equality constraints, and let's say your function looks like this. Here are some level sets of your function or something like that. So that's the optimal point. That's fine. So here are some level sets. Now if you are here, the Newton direction, I should have drawn it more skewed so the center was over there, but I didn't sorry.

So the Newton direction is probably going to want a point like that, unconstrained, but you can't because you have to stay, well you have to have something in the null space, and you have to be feasible, so it's going to push you in that direction. What's really odd is the following; you can actually imagine that you are here. Now when you're here, you've got serious problems, because for one thing, you're not feasible. The constraints are $AX = B$, you don't have $AX = B$ here, because you're off this thing.

Your second problem is that you're not optimal here, right? So you can actually define a Newton step off here, and it's actually going to combine two things. One is kind of an urge to return, I'm anthropomorphizing it but that's okay, an urge to return to feasibility or to approach feasibility, that's No. 1, and No. 2 simultaneously to decrease f, okay. And by the way, if you are here, you could actually easily be in a situation where the urge to return to feasibility will overwhelm, and you'll actually have an increase in f.

That would be clear, for example, suppose I was at the unconstrained minimum here. Well there's actually, grad f or whatever is zero, there's no direction you can go in and make f smaller, that's obvious, right? If you are here, however you're not feasible. So in this case, you're going to point towards this line here, and that of course will mean that it's going to be an ascent direction in an f. But you have, because you're not feasible. Although you have a mighty fine function value there, it's not feasible. Okay, you have this idea of the Newton step when you're off here. By the way, the Newton step will be such that if you took a full step it will always get you back to the line exactly. That Newton step is described this way, and you'd put here, this is in fact our primal, that's the primal residual.

Note that once you're feasible, if X were feasible, this would be zero, and this would revert to the standard Newton direction. But if you're not feasible this second part down here, which is a zero when you're feasible, actually contributes to your direction, and it's a direction that points you towards feasibility. In fact, it points you to a point where if you took a step of one, you would be feasible. Okay, I think we'll continue this from here next time.

[End of Audio]

Duration: 76 minutes