ConvexOptimizationI-Lecture01

Instructor (Stephen Boyd):Okay. Welcome to 364a, which is convex optimization one. Unfortunately, I have to start class by apologizing. If you're watching this, I guess you'd say – if you're watching this at the scheduled time of the class, you'll notice that I'm not there. That's because I'm in India. You can actually get very confused doing this.

There's actually one more confusing thing than this, and that's when you tape ahead of class before it actually – it goes after a class you haven't given yet. That gets very complicated. I'm sorry to miss the first day of the class. As I said, I'm in India or will be, depending on where the time reference is. That's very tacky. I've never done it before, so this is a first.

Actually, we're going to make a lot of firsts today. I believe we may be making SCPD history because we're taping ahead this lecture not only the quarter before but the year before. A lot of people are very excited about this. If this works out well, who knows where this is going to go. I could teach all of my spring quarter class the last few weeks of winter quarter. We'll see how this works. We're possibly making tape ahead history here.

A couple things I should say about the class, other than apologizing for not being present or, for that matter, in the country for the first day of it. I'll say a couple of things, and then we'll get started. For those of you watching this, since you're not going to get to see the audience and know that I sent out a pitiful plea to come if you're around for the tape ahead, you should know the room is not empty.

Let me say a little bit about the class. I think some of it is kind of obvious. We barely got the website up and running. We change it every day. Let me say a few things about it. The first is the requirements – there is going to be weekly homework, which if you are interpreting this relative to January 8, I think the correct statement would be "has already been assigned." If you're interpreting it in real time, it's actually fall, so we haven't even thought about homework one, but we will. It'll all be ready to go.

There'll be weekly assignments, and there will be a take home final exam, which will be the standard 24-hour take home exam – the usual format. You'll do serious work. The other requirement for the class – it's not really a requirement, but there will be minimal Mat-Lab programming. It's not a big deal, but there will be Mat-Lab programming, and that will be done using a system called CVX, which if you go to the website, you can find out about.

You're welcome to install it and read the user guide now. There's no reason not to. You'll be using it throughout the quarter. You might as well look at the user guide, install it, see if it works. Not everything in the user guide will make sense until you get to week four of the class, but there's no reason you can't start poking around with it now. When we get to it, you should be extremely grateful for the existence of this stuff, because it trivializes the “programming” that we’ll do in the class. Programming is in quotes, and the reason is we’re talking things like ten lines. It’s not a big deal.

By the way, on that topic, I would say this – if there’s anyone that – CVX is based on Mat-Lab. That’s the way it is. If there’s anyone who wants to try to use a real language, like Python, I will be very happy to support you in that. The TAs are Yung Lung, who I guess you can’t see, and Jacob Mattingly, who’s in New Zealand, but will not be in New Zealand when this is played back. Jacob would be very happy to – if there’s one person who does all the numerical stuff in Python. I’ll just say that, and feel free to come forward and let us know.

The only other thing I would say is something about the prerequisites. The prerequisites are basically a working knowledge of linear algebra. 263 absolutely – it’s clearly the perfect background. If not, what we mean is a working knowledge, so not the ability to know about these things but actually how you use them in modeling and things like that. The other one is the mathematical sophistication level is going to be a jump up from 263. This is a 300 level class, so the mathematical sophistication is going to jump up.

The cool part is the mathematical sophistication jumps up but actually is still in some sense a very applied course in the sense that we’re not just going to talk about optimality conditions. You will do portfolio optimization. You will do optimal control. You will do machine learning. I mean actually do numerical stuff. It won’t just be stupid programming. It’s going to be figuring out what’s going on. It won’t be obvious. It’ll be good stuff.

I’m trying to think what else to say before we start in on the class. I’ll just set it on context how it’s different from 263. 263, which is a linear dynamical systems class – I would call that basic material. It’s useful in tons of fields. It’s very widely used. A lot of people know it. It’s used in economics, navigation – all of these areas. Control and signal processing is basically all done this way. Communications – a lot of stuff is done this way. It’s a great first look at how these mathematically sophisticated methods actually end up getting used.

364 revisits it. It’s quite different in a couple ways. The first way is you might even just say this is 263 beyond least squares. In other courses similar to that, so in your first look at statistics and your first course on statistical signal processing or whatever – it’s the same sort. Everything is Galcean. All distributions are Galcean. All objective and constraints are quadratic. You do this because there’s analytical formulas for it. Behind the analytical formulas, we have very good computational tools to give computational teeth to the theory.

Here, it is going to be way beyond that. We’re going to have inequalities. We’re going to have all sorts of other interesting norms and other functions. It’s a much richer class of problems. These are problems much closer to a lot of real problems, ones where – you don’t have the concept of an inequality in 263. You have no way to deal with noise with a [inaudible] distribution in your first class on statistical estimation. We're going to look at things like that.

I should also say that the number of people who know this material is relative to 263 very small. It's probably a couple thousand. It's growing rapidly, but it's just a couple thousand. That means you're going to move from material – a lot of the 263 material is kind of from the 60s and 70s. It's not stuff that's new. This class, in contrast – you're at the boundary of knowledge, which makes it fun. Maybe you'll even push the boundary. I hope you do, since that's the point of the class to train you to find your corner of the boundary and start prospecting.

I can't think of anything generic to say, so maybe we'll actually start the class, unless there are any questions.

Student: If you haven't taken 263, is it a big problem if you have a working knowledge of linear algebra?

Instructor (Stephen Boyd):No, no problem. What'd you take instead? No problem. I should also say this – something about the background. The class is listed in electrical engineering. I'm not actually sure why. The last I checked, that's the department I'm in, so it seemed like a good idea all around. You don't need to know anything about electrical engineering – in fact, we'll talk about lots of applications throughout the quarter, and honestly, probably for half of them, I don't even know what I'm talking about. That will happen. You'll be in some field. I'll super oversimplify it. You're welcome to point it out.

The point is, I'll talk about circuit design. We'll talk about machine learning. On all of these topics, there will be people in the class who know more than I do about it, and there will be a lot of people who don't know anything about it. If you're one of those latter, don't worry about it. They're just examples. There is one prerequisite, although I don't know what would happen if you didn't have it. That is that I will assume that you know about basic probability estimation and things like that.

Do I have to say anything about the textbook? Go to the website. That's all I have to say. There's a textbook. You'll find it at the website. I'll start by covering broad basics. It's not in any way representative of what the class is going to be like, what I'm going to talk about today. Don't think it is. We're going to cover the basics, and maybe I'll get into something that's real. This is going to be the very highest level. We'll talk about mathematical optimization. We'll talk about least squares and linear programming, convex optimization. We'll look at a very simple example, and I'll say a little bit about the course goals and style.

Let's start with this. Optimization. Depending on the company you're in, you may have to put a qualifier – mathematical. If you say just optimization, it goes to something on complier optimization or something in business practices. If you're in a broad crowd like that, you need to put mathematical in front to make it clear.

Here's what it is. You want to minimize an objective function. You have a variable X with N components. These are loosely called the optimization variable. Colloquially, you would say that X1 through XN are the variables. By the way, there are other names for these. People would call them decision variables. That's another name for it. In a different context, they would have other names. Decision variables is a nice thing because it tells you these are things you have to decide on.

When you make a choice X, you'll be scored on an objective. That's F0 of X. We'll choose to minimize that. In fact, you ask what if you wanted to maximize it? Then you would just minimize the negative of that function, for example. It's convenient to have a canonical form where you minimize instead of maximize. This will be subject to some constraints. In the simplest case, we'll just take a bunch of inequalities – that's FI of FX is less than BI. Here, without any lost generality, the BIs could be zero, because I could subtract BI from FI and call that a new FI. There'd be no harm.

This is a nice way to write it because it suggests what really these are. In a lot of cases, these are budgets. It's a budget on power. It often has an interpretation as a budget. There are other types of constraints we'll look at. What does it mean to be optimal for this problem? By the way, this is redundant. You should really just say an optimal point or the solution. In fact, you shouldn't trust anyone who says optimal solution, because that would be like people who say the following is a true theorem. It's kind of weird. It's redundant. It doesn't hurt logically, but it kind of – what about the types when you forgot to add the attribute true? What were those theorems?

I'd say an optimal point or a solution is a point that satisfies these inequalities here and has smallest objective value among all vectors that satisfy the constraints. That's what you mean by a solution. Of course, you can have everything possible. You can have a problem where there is no solution. You can have a problem where there's one, where there's multiple, and you can have other things, which we'll get to more formally. This is just our first pass.

Let's quickly look at examples. Here are some absolutely classic examples from all over. The first is portfolio optimization. In portfolio optimization, the decision variables X1 through XN – these might represent the amount you invest in N assets. For example, if XI is negative, it means you're shorting that asset, unless that asset is cash, in which case you're borrowing money. The X1 through XN is actually a portfolio allocation.

You'll have constraints. Obviously, there will be a budget constraint. That tells you the amount that you have to invest. There might be maximum and minimum investment per asset. For example, the minimum could be zero. You're not allowed to actually purchase a negative quantity of an asset, which would be shorting that asset. You can have no shorting.

You might have a maximum amount you're allowed to invest in any asset. You might have a minimum return that you expect from this portfolio. Your objective might be something like the overall risk or variance in the portfolio. You'd say here's 500 stocks you can invest in. My mean return absolutely must exceed 11 percent. Among all portfolios that meet – I'm going to have no shorting, and the sum of the XI is going to equal one because I'm going to invest one million dollars.

The question is among all portfolios, they guarantee a mean return of 11 percent. I want the one with the smallest variance in return. That would be the one with the least risk. We'll look at these in detail later. This is one where when you solved the optimization problem, it would probably be advisory in the sense that you'd look at what came back and say well, how interesting. Maybe I'll execute the trades to get that portfolio or maybe not. This could also be real time if you're a hedge fund and you're doing fast stuff. This could be programmed [inaudible]. There are lots of different things this could be used for.

Let's look at [inaudible] electron circuit. Here, you have a circuit. The variables, for example, might be the device widths and lengths. It could be wire widths and lengths, something like that. You have constraints. Some of these are manufacturing limits. For example, depending on what fabrication facility is making your circuit, you have a limit on how small the length of a gate and a transistor can be. It can't be any smaller than 65 nanometers. That's the smallest it can be.

That's a manufacturing limit. You would also have performance requirements. Performance requirements would be things like timing requirements. You could say this circuit has to clock at 400 megahertz, for example. That tells you that the delay in the circuit has to be less than some number because it has to be done doing whatever it's doing before the next clock cycle comes up. That's a timing requirement.

You might have an area requirement. You could say if I size it big, I'm going to take up more area. This portion of the circuit can't be any more than one millimeter squared. The objective in this case might be power consumption. You'd say among all designs that can be manufactured in the sense that they don't reject your design instantly and meet the timing requirements among all of those, you want the one or a one with least power. That would be minimized power consumption.

Our last one is just from statistics. It's a generic estimation model. Generic estimation looks like this. What's interesting here is these are from engineering and finance and stuff like that. In these cases, the Xs are things you're going to do. In this case, they're actually portfolios you're going to hold, and in this case, they will translate into polygons that get sent off to the fabrication facility. The [inaudible] are actions. Here, it's very interesting, because the XIs are not actions. They're actually parameters that you're estimating.

Here, you have a model. You can take any kind of data or something like that and you'll have parameters in your model. You want to estimate those parameters. These XI are not things you're going to do. These are numbers you want to estimate. That's what it is.

You have constraints. For example, let's say that you're going to estimate a mean and a covariance of something. It can be much more sophisticated, but let's suppose that's the case. We have a constraint here. There might be some constraints. Here's a constraint.

The covariance matrix has to be positive semi definite. That's a constraint, because if you created a covariance matrix that wasn't, you'd look pretty silly. That's a nonnegotiable constraint.

You could also have things like this. You could say these Xs must be between this limit and that limit. For example, suppose you're estimating some diffusion coefficient or some parameters known to be positive. Then you should make it positive. These are parameter limits. The objective is, generally speaking, dependent on how you want to describe the problem. If you describe the problem in an informal way, it's a measure of misfit with the observed data. For example, if I choose a bunch of parameters, I then propagate it through my forward model and find out what I would have had, had this been the real parameter.

The question is, does it fit the observations well? Another way to say it – it's a measure of implausibility. That's really what it is. It's a measure of implausibility. In this case, we're minimizing it. In many contexts, you'll see it turned around so it's maximized. If you're a statistician, you would reject the idea of a prior distribution on your parameters, and your objective would be to maximize the likelihood function or the log likelihood function. That's what you'd be maximizing. That's essentially a statistical measure of plausibility. You'd minimize the negative log likelihood function, which I think they call loss in some of these things.

Implausibility in a [inaudible] framework, a measure of implausibility would be something like the negative log posterior probability. That would be a measure of implausibility. If you minimize that, you're actually doing MAP, which is maximum a posteriori probability estimation. By the way, we'll cover all those things again, so this is just a very broad brush. If you don't know what these things are, you will, if you take the class.

These are examples of optimization. Everyone in their intellectual life goes through a stage. It can happen in early graduate school, mid graduate school. It can also happen in later life, which is bad. It's not good to have it when you're an adult. Let me describe this stage of intellectual development. You read a couple of books and you wake up at 3:00 in the morning and say oh my god, everything is an optimization problem. Actually, a lot of books start this way. My answer to that is – you have to go through this stage, so that's fine. But get over it quickly, please. Of course, everything is an optimization problem.

What you'll find out quickly is it doesn't mean anything to say that. It says nothing. What matters is what optimization problem it is, because most optimization problems you can't solve. They don't tell you that. Typically, they don't tell you this. Or, what they do is they do tell you, but they distribute it through the whole quarter. It turns out if you just say a little bit of that message every day, at the end of the quarter, no one can accuse you of not having admitted that we don't know how to solve most optimization problems.

However, because it was below the threshold of hearing in each lecture, as a result, all these students went through and the big picture didn't come out, which is basically – the way you cover it up is by going over 57 different types of algorithm for solving things, which basically is a cover up for the fact that you can't solve anything. We'll get to that.

This is related to the following – solving optimization problems. To say that everything is an optimization problem is a stupid tautology. It all comes down to this. How do you solve them? It turns out it's really hard, and basically in general, I think it's fair to say it can't be done. I can write down shockingly simple optimization problems and you can't solve it, and it's very likely no one else can. I can write down very innocent looking optimization problems and they'll be NP hard.

What do people do about it? Well, there are a couple of ways to do it. What do I mean by NP hard? Nondeterministic polynomial ties. You take at least quarters on this in a computer science class. I'm going to give you the 15-second version. I'll tell you about it from a working perspective. In the 70s, people identified a bunch of problems that so far no one had figured out a good polynomial time algorithm for solving.

Polynomial time means there's an algorithm where the problem comes in, you measure the size by the number of bits required to describe – you measure it by the number of bits required to describe the data, and then you look at the number of operations you have to do. If you can bound that by a polynomial, then you'd say that's polynomial time. Bear in mind, I'm compressing a ten week course to about 14 seconds. I'm going to gloss over some of the details.

There were a bunch of problems where people – the famous one would be traveling salesmen problem. No one had found a polynomial tying method. A guy named Cook, maybe, did the following. He catalogued a bunch of problems and said if you can solve any of these, then if you make a method for solving that, I can make a reduction. I can take your problem and map it to this, and with that, I can solve the traveling salesmen problem. Then you had an intelligent way of saying of two problems, one is just as hard as the other.

NP hard means, and this is really – people are going to cringe if they have a background in this – it means it's at least as hard as a catalog of problems thought to be very hard. That's your prototype. Basically, what I'm saying is not true, but as a working definition of what it means – for your purposes, this is going to be fine. It means the problem is at least as hard as traveling salesman.

Let me tell you what that means. It is not known that these things cannot be solved from the polynomial tie. That's not known. It is thought that that's the case, and it may be at some point, somebody will show that you can't solve these. Right now, they're thought to be harder. I think there's an insurance policy. Let me tell you why it's an insurance policy. A ton of super smart people have worked on all these problems, and now all of these things are banded together as you would do in insurance. You band a whole bunch of things together.

What would happen is if tomorrow, somebody, probably a Russian, proves P equals NP, meaning you can solve all of these problems, it would indeed be embarrassing for a lot of people. However, the embarrassment is amortized across – people could say you went around and made a big deal about convex problems and polynomial time. I say look at all the other people. The embarrassment is spread across a great swath of people. It's an insurance policy. It is thought that they're really hard. If they're not, you're in very good company with the people who also thought they were hard.

Student: It's just in its own field. It's not proven to be certain. It's not mathematically proven to be non-polynomial.

Instructor (Stephen Boyd):That's right. That's why it's still an open conjecture. If, in fact, it turns out that these are theoretically not hard, the [inaudible] could end up being huge, and that would also blunt the embarrassment. In any case, the embarrassment is spread across a lot of people. We'll come back to that problem several times in the class.

Methods to solve a general optimization problem – they always involve a compromise. The two main categories of compromise are as follows. The first one is to not back down on the meaning of solve. Solve means solve. It means find the point that has least objective and satisfies the constraints. That's the definition of solve. You leave that intact, but you leave open the possibility that the computation time might involve thousands of years or something like that. People call that global optimization, and it's a big field. It is almost entirely based on convex optimization.

The other one, which is the one most widely done by people who do optimization – they do the following. It's very clever what they do. They go and solve – they put a footnote, and then way down at the bottom, they write this – they write and now I'll have to ask you to zoom in on that. There it is. It says not really. What that means is they make a big story about this, and they say it's a local solution. What happens is they don't – it gets tiresome to say solve locally, plus, it doesn't look good.

What happens is you drop it after awhile, and then you say I solved that problem, and people give talks and say I minimized that. The point is they didn't. What they did was they got a local solution. We'll talk more about that as well. There are exceptions. If you were going to do that, I would maintain, although this is not widely held – you'll find that many of my opinions are not widely held. If you were going to do that, I would do it via convex optimization.

Student: But you will be doing it in the spring?

Instructor (Stephen Boyd):Yes. It's scheduled to be in the spring. Who knows when I'll do it. What are the exceptions? There are cases where you can solve these problems with no asterisk. The most famous one in the world by far is least squares, least norm. No asterisk there. It's not like oh, yeah, well, I transpose A inverse A transpose B. Yeah, that often works super well in practice. The status of that is that is the global solution. There are a couple others that you might not know about, and that would be things like linear programming problems. We'll get to those quite quickly. Those are ones where you get the answer.

There are asterisks in these, but they're much smaller fine print. I'll get to them later in the class. When there's an admission that has to be made, I will make it. The parent of these, and a considerable generation, is this convex optimization problem. This is really the exception. The rough idea, to put it all together, is something like this. Most optimization problems – let's review what we've learned so far. A – everything is an optimization problem. B – we can't really solve most optimization problems. The good news is here, that there are huge exceptions. This entire class is about one of the exceptions.

When you do convex optimization, there are no asterisks. You solve the problem and there are no apologies. It's the global solution. Is life convex? I would have to say no, it's not. I hope it's not. It's not sad. If we get to that later today, you'll know why it's not. To check convexity, you have to take two instances and then form weighted averages in between. What would the average of yours and your life look like? The other thing that has to happen is that life has to be better than the average of the qualities of your lives. Let's keep that as a running topic that we'll revisit periodically. For the moment, my position is it's not.

Student: Are there other exceptions besides convex optimization?

Instructor (Stephen Boyd):Yes, there are. Singular value decomposition. That's one where our can compute the singular value – I can write it out as an optimization problem pretty easily. I could say find me the best rank two approximation of this matrix. That's way non-convex. Yet, we can solve it and we get the exact global solution. That's an example. There are some combinatorial problems. So if you've taken things on combinatorial algorithms in computer science – combinatorial algorithms on their face would appear to be non-convex. It turns out a lot of them are convex. It's quite strange.

You take something that's a combinatorial optimization problem that on its face would not be. It turns out if you let the variables vary between zero and one, you can prove that there's always a solution, which is on the vertices, and so there's actually a lot of problems that we can solve but are not convex. Some of them can be secretly turned into convex problems. Getting a rank two approximation of a matrix is an excellent example. We can definitely do that and it is definitely not convex.

We have least squares or, if you're in statistics, regression. It may have other names. I don't know. Here's the problem. You want to choose X to minimize the [inaudible] norm squared of AX minus B. If A is full rank or skinny, you get an analytical solution, which you know if you know about linear algebra. It's just A transpose A inverse A transpose B. In this case, it's a unique solution. In fact, we have a formula for it, which is overrated. In this class, we're going to look at tons of problems. There will be analytical formulas to almost none of them. You'll have to wean yourself away from analytical formulas.

The sociology of that is very interesting. You've been trained for 12, 14 years on 19th century math, which was all based on analytical formulas, but we're going to move beyond that. If you look most deeply into what it means to have an analytical formula, it turns out – we're going to solve AX minus B with an infinity norm there or a one norm. There's no analytical formula for it now. But it turns out we can calculate that in the same time it takes for you to calculate this. Having a formula is something that will mean less to you by the end of this class.

Student: So not all optimization problems have that subject.

Instructor (Stephen Boyd):When we go over this in hideous detail later, that's the case. I should mention you should be reading chapter one and chapter two. Chapter one will have a lot of this vague stuff, and chapter two will start in on the real stuff. You're absolutely right. An optimization problem – you do not have to have any constraints, in which case it's called unconstrained, and you don't even have to have an objective. If you have neither, that's called a stupid problem. It's minimized. The universal solution – X equals zero.

If you don't have an objective, it's called a feasibility problem. In some fields, they call it a satisfaction problem. You have a bunch of inequalities and you want to find a point that satisfies all of them. That's what that is. Back to least squares. Much more important than the formula – it turns out you can write down a formula for a lot of stuff, and it doesn't do you any good if you actually want to calculate it. Here, there are reliable and efficient algorithms that will actually carry out this computation for you.

By the way, that's why least squares is so widely used because it has computational teeth. Instead of just talking about it, which is what a formula would allow you to do, you can actually use it. You can actually compute. We're not going to go into too much detail in the numerics. At one point in the class, we will. We'll talk about how you exploit structure and things like that. Here, just so you know, the computation time to solve a least squares problem goes N squared K. N is the number of variables, and that's the small dimension. K is the number of rows in A. It's a good thing to know. It's the small squared times the big.

You can do this as I said not long ago in 263 with modern computing. It's amazing what you can solve. Then, we did a couple thousand row least squares problem. You can call it a regression problem in 2,000 dimensions with 1,000 regressors. It was three seconds. By the way, if A is sparse or has special structure – suppose part of A has an [inaudible] embedded in it. That would come up in medical imaging. You can do that faster. In image processing, you'd have transformations. You can solve least squares that are way bigger.

I would say that least squares is a mature technology. When I do this, people who worked on all of this – it's a huge, active field in lots of areas – they get extremely angry when I use the word technology. I said by the way, I mean technology here. This is the highest praise. This is not an insult. What it means is that other people know enough about the theory, the algorithms, and the implementations are so good and so reliable that the rest of us can just type A backslash B.

Of course, if you're building some real time system or the sizes get to a million or ten million, you're not going to be using backslash. But that's it. That's a boundary that's growing with time. That's the wonderful thing about anything that has to do with computers. Just take a vacation for a year. My colleagues at the other end of the [inaudible] who actually do real things, they and all their friends around the world will make stuff twice as fast. You just go away. You go to the Greek Islands for three weeks. You come back and computers are faster. It's great.

Of course, I'm not telling people to just use A backslash B. Everyone here has done A backslash B. Probably only a few know what it actually did. Nothing terrible has happened. I'll come back to them and when they're yelling at me, I'll say back off. Do you use TCP/IP? Sometimes, they won't even know what I'm asking. Then I'll say are you using TCP/IP as a black box? You don't even know what's inside it?

Some of them will say yeah, I do. Even communications on your laptop between different components – I'll say do you know what it does? Most of the time, they'll say no. Here's what you need to know. It is not trivial by any means to make a reliable bit pipe to transfer bits from one place to another with unreliable medium insight. It's no more trivial than it is to solve this problem numerically. It is not trivial. All you need to know is this. Very intelligent people have thought about this problem very deeply. They've thought about what can go wrong, deadlocks, all sorts of crazy stuff, and they have come up with algorithms about which they know an incredible amount.

Part two – other people have implemented them, and these are very reliable implementations. The result is for most of us, we can just consider certain things to be reliable bit pipes. We don't have to care about how many packets were retransmitted or how the flow control went and all that kind of stuff. Like least squares – if you're doing real time control or something like that or if you're doing some computing that requires extreme reliability, then you can't treat TCP/IP as a black box.

You might ask does this calm them down? The answer is no. This makes them more angry. I mean technology here in praise of this. When you use least squares – if you've just come from 263, you've used least squares. That's just the beginning. The way you use least squares is this. It's easy to recognize a least squares problem. Sometimes, it comes to you on a platter. Somebody walks up to you and says I took these measurements. It's a linear measurement model. Help me guess these parameters. I received a signal and went to this channel – this kind of thing.

If you learn a few tricks in least squares, you can do well. If you learn how to do weight twiddling and you learn about regularization, those two alone – you are now trained and ready to do battle with using least squares as a tool. You will do really well. Weights is basically you go into a problem and someone says there's no limit on the sum of the squares. The limit is on this. You say no problem. I have these weights here. You look at the least squares solution. You don't like what you see. You change the weights and you do it again.

In engineering, we do this all the time. It's called weight twiddling. It's a slightly derogatory term, but not bad. I'm sure that you do it in statistics, but I don't know that they admit it. Good. When I'm making fun of a department, I like to have a face to look at. They like to stick to – if you're doing real statistics, you go back in and change the prior distribution. I have to warn you, all of these lectures are being put on the web. It's weird and fun. We'll fuzz out your voice, and if the camera focused on you, we'll put the squares on it. Don't worry about it. If you'd like, we can obscure your department. We can beep it out.

I'm just going to guess that in statistics, they make a big story about the prior distribution. I bet if they don't like the way it comes out, they go back and change that prior distribution. They cover up their tracks. We do it in engineering, and we're not embarrassed. I bet they do it. Next topic is linear programming. Some of you have seen this. How many people have seen linear programming somewhere? A bunch. Okay.

Linear programming – in my opinion, it's what people should learn immediately after least squares, singular value decomposition, linear programming. If you're going to stop somewhere, that's a very good place. I'm taking about if you really want to go out and field algorithms, do real stuff – that's a great place to stop. Everybody should know about it. If you take this class, you're going to know a lot about it. Linear programming is a problem that looks like this. Minimize a linear function subject to a bunch of linear inequalities.

We're going to talk about it in horrible detail later in the class, so I'm not going to go into too much detail now. I want to talk about the attributes of linear programming. The first is in general, except for very special cases, there's no analytical formula for the solution. There is no A transpose A inverse A transpose B. By the way, don't confuse that with our inability – you can say a huge amount of qualitative, intelligent things about a linear program. What you can't do is write down a formula for the solution.

There are reliable and efficient algorithms and software. In fact, as a homework exercise, you will write something in the top class linear – it will be 50 lines of Mat-Lab or Python, and you'll write something that will solve huge linear programs quite well. It's no asterisk on solve – you get the solution. The computation, by the time, is proportional to M squared N. That's exactly the same as least squares. If you equate rows of the least squares objective with constraints, it's identical. That's not an accident.

M is a number of constraints here, and K is the height of A.

Student: So this M – there's still that many rows in A.

Instructor (Stephen Boyd):Yes, if I write that as a matrix inequality, AX less than B, yes, that would be – this M would be that K. We spent hours discussing whether this should be M or K, and we finally – it probably should be K.

Student: What about C?

Instructor (Stephen Boyd):X is an RN, so C is an RN, too. Actually, linear programming – it's a very old topic. Fourier wrote a paper about it. The modern era starts in the 30s, where else but Moscow. The modern era traces back to Stanford and George Dantzig. LP was something that you just talked about until you had computers, at which point LP looked a lot more interesting. That was 1948. I think a lot of people knew about linear programming. Something like this coupled with computers – that's a mature technology.

Linear programming – it looks like it would be easy to recognize, and in some cases, a problem really comes to you on a platter like this. Someone comes to you and says help me solve this problem. I want to minimize this weighted sum of the variables and I have some budget constraints. There are three problems that have exactly this form. Here's the really cool part about linear programming. You will be stunned later in the class when you see the kind of problems that we can reduce to linear programming. Things that do not look like linear programming at all we will reduce to linear programming.

What that means is they're solved. Unless your problem is huge or you have some super real time thing like in communications, then there's a sense in which you're kind of done at that point. If you do medical imaging, that's a mistake, because the problems are too big. You can't say it's a linear program and walk away. You can't do communications. It depends on the time scale. You had to adapt to things. You can't detect the bits transmitted in a packet, because that's coming at you way too fast. I would recommend that you go into fields that are in between in size.

We're going to see a bunch of tricks. Finally, I'll say what convex optimization is, because it would seem in a class with that title, one should say what it is in the first lecture. Here's what it is. It's an optimization problem – minimize an objective subject in constraints. Here's what has to happen. The function F0 and the FIs have to be convex. You know what convex is. It means that the graph looks like that. That's a convex function. The graph should have positive curvature.

Least squares problem has that form because if I look at the least squares objective and I look at the plot of it, it's a quadratic function squared, and basically, it looks like a bowl. If you take a slice at a level set, you get an ellipsoid. It's convex. Linear programming also is a convex problem because all of the objectives are linear. Linear functions are convex. Linear functions are right on the boundary. They have zero curvature.

One way to say convex is just positive curvature. This includes least squares, and kind of the central idea at the highest level of this class is this. If you want to solve a convex optimization problem, there are no analytical solutions. There are in special cases. We'll see tons of cases in communications and various other places where they have special analytical solutions. You've seen one in least squares already.

In general, there isn't an analytical solution. However, there are going to be reliable and efficient algorithms for solving these with no asterisk. You will get the solution. In fact, if someone came from another planet and landed here and asked you what you're doing, there would be – it would be very difficult to make an argument that solving a convex optimization problem compared to a least squares problem was, for example, that you'd been reduced to a numerical solution, which is what a lot of people might say with a hint of – they don't like the idea. They say numerical method in a way that makes you want to go wash your hands.

The computation time for solving convex problems is roughly proportional to something like N cubed – the number of variables cubed – N squared M and F, where F is the cost of evaluating the functions and their first and second derivatives. We don't have to get into that, but the point is it's like least squares. You can basically solve these. You will solve them in this class. You'll know how to write the algorithms to solve them and stuff like that. It is an exaggeration, in fact, to say it's a technology. It's almost a technology, and every time I give this class, it's getting closer and closer.

I should say something about – we'll get to it later, but this is a very profound statement. It basically identifies a huge class – let's review what we know so far. A – everything is an optimization problem. B – we can't solve any optimization problems despite ten or twenty weeks of lectures on it. Now what I'm saying is on the positive side. It's really cool, and it's not obvious at all. It says if the objective and the constraints all have positive curvature, then this thing is just like least squares. You can solve it exactly. You can solve it quickly. Although I'm not going to focus on it, you can say a whole lot about it theoretically.

We will say some stuff about it theoretically, but that's not going to be the focus of the class. This is a pedantic way. You might prefer it if I wrote it this way. Four Theta N zero one. You might prefer it that way.

Student:I don't know why it has to be zero one, though. Why can't it be 99 and 100.5 or negative five?

Instructor (Stephen Boyd):There are versions where there are different constraints. If I just say – it said F of Alpha X plus Beta Y is less than or equal to – if it's Alpha F of X for all Alpha and Beta, the function would be called sub linear. It's a different thing. This is just a definition of convex.

Student: It doesn't have any physical basis or anything. It's not gonna turn concave if you suddenly make it more than one.

Instructor (Stephen Boyd):We'll answer your question. This is not an accidental. For now, that's the definition.

Student:[Inaudible] just a statement of the [inaudible] of the line between –

Instructor (Stephen Boyd):That's what it is. It's this picture. By the way, we're going to get to this later, so you're not supposed to be understanding everything. First of all, I'm not saying that much. The stuff I am saying is kind of vague. You're not supposed to be parsing this. You're supposed to have your relaxed parsing mode on and let me get away with stuff. There'll be a time for is that really a minus sign. That's coming later.

You already know something you didn't know. Optimization problems where the objectives and constraints have positive curvature, roughly speaking, we can solve. The theoretical implication, I think, is extremely interesting. The practical ramifications of this are absolutely immense. It means you're messing on some problem in your own field, and if you get it to a problem of this form – it won't be obvious.

It's much cooler when you get to it and you turn things around and you switch some variables around. All the smoke clears. There are some functions there that it's not remotely obvious they're convex. You show they are, and then you're a hero, and it's fun. It means you can now solve those problems.

I give a talk about these things lots of places. People say that's really cool. How do you learn about this? How do you know if a problem is convex or not? I go no problem. You just have to come take my class. You do ten homeworks, each of which takes 20 hours and then you do this 24-hour final. After that, for sure you'll be quite well trained. Then they're less enthusiastic about it.

It's actually often difficult to recognize convex problems. That's going to be a theme of the class. Let me distinguish a few things there. I would distinguish problems that come on a platter convex, the ones where you have to do some work and transform them and stuff like that. Let me move on.

I want to do an example just to give a rough idea of what this looks like. For people who did 263, this will kind of tie into that. Here's our problem. I have a surface here with patches, and I have some lamps here. We're going to choose the illuminating powers on the lamps. That's going to be our variable.

The illumination on a patch here is going to be the sum of the illumination from all the lamps here. It's going to be proportional to the lamp power and then the proportionality constant is going to be an inverse square law. R is the distance between the patch and the lamp. There'll be something which is a shading effect, because obviously, if you're coming in straight on here, then you're going to get the full power.

If this lamp, for example, puts very little illumination on here, and this were below its horizon, if there were a lamp here, it would not illuminate this patch at all. This is a simple thing. The problem is to achieve a desired illumination which is given. You want to do this with bounded lamp powers. I have a maximum power. Powers cannot be negative. I care on a log scale, because what I care about is missing the lamp power by two percent or twelve percent. I don't care about absolute. It's ratio compared to this one.

The question is how would our solve it? Let's take a look and see how you might solve it. If your question is do I shamelessly reuse material from one place in another, I can confirm that, yes. Are you asking have you seen that figure before? The answer could well be yes, you have.

Let's talk about how to solve it. The first thing you might do, and I would recommend this – the first thing you do is you say let's try some simple suboptimal schemes. One is just this – you set all the lamps at the same power. You vary it and you plot this objective. You do a search over it. There's no dishonor in it. You do that. That might work, but it might not. That's a good baseline design. You could say, well, I just learned least squares from 263. You're going to use least squares. I'm going to do this.

The objective here is not the sum of the squares. This is real life. This is the illumination engineers. Everyone uses the absolute value – the sums of the absolute values of the law of percentage error. I'm making it up, of course. You say well, good for you. We use the sum of the squares. You solve this problem. When you solve this problem, I guarantee you some of the Ps are going to come out negative. By the way, you're going to do super well. You're going to get a nice, flat illumination pattern.

What will happen is you'll look at the powers and a whole bunch of them will be negative. It turns out you can do really well in illumination with lamps that can spray negative power out. That's not good. Then the heuristic. What do you do? Here's what you do. You say if a P is bigger than the maximum power, I just set it equal to P max. If it's a negative lamp, I turn that lamp off.

Once again, you see how well you did, and you might do better than uniform – maybe worse. I don't know. Now, this is what someone trained in least squares would do. They'd say not a problem. They'd go over here and say I want PJ to be in the interval zero P max. Therefore, I'm going to add a regularization term which charges P for being away from the center of that interval. Everybody see what this is?

You start with all the Ws one, and you solve this problem. Then, you just start weight twiddling. You'll do quite well here. You'll come up with some algorithm that twiddles the weights, and how you twiddle them is totally obvious. If P comes out outside that interval, that weight needs to go up in the next cycle. If P is timid because your weight is so high, you want to decrease that weight. A couple of cycles of this, and you're going to get a very, very good solution.

Unfortunately, you might also go on to then write an impossible to read 12-page paper with pages and pages describing your iteratively reweighted illumination thing. Hopefully, you won't. You could also use linear programming. If you know about linear programming, you could actually solve this L one problem where you minimize an L one norm. This is closer than that. Linear programming – it handles the inequalities. There's no twiddling there. This would probably do the best of all of them.

The problem is convex, so this objective function – it just looks like this. It's linear over here, and then it's an inverse over here. You look at that function and you realize a couple of things. The important part – that's what you're going to be trained on in the next ten weeks is looking for convexity. This is what we like to see.

By the way, you will see that that function is not differentiable. In a lot of other treatments of optimization, differentiability has this very high role, because a lot of things are based on gradiance and derivatives, and there's no derivative there. So in convex optimization, differentiability is going to actually play a much lower role. In fact, it's a non-role.

This problem, even though it is non-differentiable here, it can be solved as fast as least squares if you know what you're doing. We might even have you write from scratch a solver for it. We could also assign it. It would be four lines in a higher-level language or something like that to solve this. That's it. This is just an example of a problem where it's not obvious exactly how to solve this, and you can imagine a lot of people not knowing.

Let's look at a couple additional constraints.

Student: Where did that curve come from?

Instructor (Stephen Boyd):I plotted it.

Student:I mean the equation. How did you know to do [inaudible]?

Instructor (Stephen Boyd):If you go back and exponentiate my other objective, you'll find this.

Student: So you just did it on both sides.

Instructor (Stephen Boyd):Yes. If you minimize something, it's the same as minimizing the X of that thing because X is monotone increasing. Let's add some additional constraints here. Here's one – no more than half the total power is in any collection of ten lamps. That would be one. Another one is no more than half the lamps are on, but you get to choose which half. I won't go into all the gory details, because for one thing, I'm running out of time, but it's not – both of these sound complicated or easy or something like that. If you don't know about convexity, you wouldn't know the second one is an unbelievably difficult problem.

In fact, you'd have to check all N [inaudible] N over two sets of half the lamps and for each one, solve the problem. Basically, everything would come down to that to actually get the real answer. You could get very good heuristics that would guess an answer, but they would not be certified as the actual correct one. This one – no more than half the total power is in any ten lamps – that actually is convex, and it’s not obvious. By week three of this class, you will know things like that.

This is actually cool, because these are things that look very similar. If I said them the right way, I could probably make you think that they are kind of the same question. I’ll often do that in talks, except I don’t give the answer, and then I pick some poor person. The point is these are just not obvious at all, these things.

Why is it famous? It’s not famous.

Student:I mean, people have [inaudible] papers and have investigated [inaudible].

Instructor (Stephen Boyd):You mean the illumination problem? I think I probably made it up one day and then actually [inaudible] – I can say this because he’s a colleague of mine at MIT. The next thing I knew, it was in his book, the famous illumination problem. You’ve been subjected to it in 263. That would be the 263 version of the illumination problem.

Notice that we didn’t ask you to find any powers, because you would have had this annoying problem of negative powers coming out. That’s selective, though. I don’t think the problem is any more famous than it’s been here. I don’t know.

No, because this would subsume all – you have to choose which half of them are actually going to be possibly on, not which are actually on. It’s exponential in the end. Let me just say a little bit and then I’ll quit. The course goals and topics – the goal is to allow you to recognize and formulate problems. You should know enough to develop code for these things. You’ll see it’s very simple. You might not think that about seven weeks into the quarter, but you’d be shocked at how simple some of these things are.

We’ll focus a bit on the theory of this, and do want to skip forward to something at the very end here. I know I’m over, so I’ll just take a minute or two to do this. I’ve already mentioned this. First of all, the theory of convex optimization is more than 100 years old. The first paper on this is from 1890 or something like that.

In fact, the idea is traced earlier. By 1970, it was very well known, the theory. There are some points here that I think I don’t have time to go into, but you can read about this in the first chapter. What I do want to say is something about why it’s interesting now, if this is 100 years old as a mathematical subfield. What makes it interesting now?

What I can say is since about 15 years ago, people have been finding applications in lots and lots of areas, and it just sort of starts coming up. It may not particularly help you in some area to know that a problem is convex, but it sure doesn't hurt. It might in some cases allow you to solve it. This was sort of not the case, with the exception of least squares and linear programming.

These have been widely used since the 50s and widely applied since the 1950s. Basically since the 1990s, there were a lot of things found in areas like machine learning and statistics and control and signal processing and stuff like that. You get a nice, positive feedback loop because once more people start discovering these problems and start writing papers on, for example, the famous illumination problem – once they start appearing, what happens is people who write the algorithms see two of these and say somebody should write a code for the illumination problem.

Then, they write a beautiful code, well tested, numerically stable, hopefully open source and all that, and then everyone can now solve the famous illumination problem. Only then do people realize that there never was such a problem, hopefully. I quit here, and I'll see you, in theory in two days, but more like two months or something like that.

[End of Audio]

Duration: 80 minutes

ConvexOptimizationI-Lecture02

Instructor (Jacob Mattingley):Okay. Good morning and welcome to EE364-a. This is the second lecture, and in case you haven't guessed, I am not Professor Boyd. He is currently on his way back from India, and he'll be here in time for the lecture on Tuesday. You don't need to worry. It will be a proper professor from Tuesday onwards.

A couple of things before we move on. First of all, the first couple of weeks are going to be relatively dry mathematical material. There's a bit of ground that we have to cover before we can do the really interesting parts, so some of that starts today. As part of that, we're going to be going quite fast indeed, so hopefully lots of you have read or scanned part of chapter two, because we're hoping to get through most of that if not all of that today.

If you do find that I'm going quite fast today, that's hopefully because I am. You mind find that it's helpful if you go back and read chapter two once you're done with this lecture. Before we dig right in, another thing is that we have posted homework one. It's not due until the 17th, which is next Thursday. Homework 1's already out. Office hours for TAs have been posted, and there will probably be more to come.

Today, we're going to be talking about convex sets. I'll start off by just describing what an affine set is and then a convex set. We'll go through some examples, then we'll talk about how you make more complicated convex sets from simpler ones and talk about a few other topics towards the end. This should be review to start off. We can form from any two points – we can parameterize the line going through those two points using a parameter such as Theta. In this example, ? can vary over the whole real line, and it should trace out the whole line.

If we've got a point X1 here and a point X2 – when ? is zero, we're sitting right at X2. When ? is one, we're sitting at X1. It traces out the whole line like this. More generally, an affine set can be described as a set where any two distinct points and the line through them are all inside the set. So take any two points from an affine set, trace the line through them, and that line better be in the same set.

One example of an affine set is a solution set of linear equations, which you've all seen by now. There are lots of different ways of writing it. We can write it with a [inaudible] parameter. We can write it using an equation on the left-hand side here, but at the end of the day, this is one of the ways you can represent any affine set. A convex set, which is going to be of considerable interest to us this quarter, is one where instead of being able to very theta over the entire real line, we're instead constrained to having ? be between zero and one. It can take on any value between zero and one.

If we find two points in our set, instead of before where we could have any point on the line through these two points, the line segment described with ? here has ? varying over zero, one. So to start off with, line segment ? between zero and one and these are the points traced out by ? as we vary it.

The convex set is a set which takes any two points in the set – so take X1 and X2 from your convex set, and the line segment between those two points had better be in the set. That means that an affine set is, in fact, a convex set, because an affine set contains the entire line through X1 and X2. Of course, it does contain the line segment between X1 and X2. Our first example of a convex set is the affine set from before.

This is just a formal way of expressing it here. We take X1 and X2 out of this convex set C. We vary ? between zero and one, and the convex combination of points here has to lie in the set C. Here are some examples. The set here – obviously, if I had picked these two points here, every point along this line lies inside the set. This here is a problem, because if I pick these two points here, all of these points here do not lie in the convex set. That clearly violates this set membership here, and this cannot be a convex set.

This one here is a more technical example. This one looks like it should probably be convex but for some of the points where the boundary is missing. If I had picked a point here and a point here, this line actually is not inside the set, because it's only got a partial boundary around here. These are some very simple examples described graphically constrained to just our two. You'll see quite quickly that convex C can make sense in quite bizarre spaces.

Following on from convex combinations, before, we had just two points involved in our determination of whether a set was convex. The thing is that we can actually extend this concept to a convex combination. If we picked points X1 through XK to start with, we can describe a convex combination of those points as being a set where we weight each of the points, X1 through XK, and the weights all have to be nonnegative and they have to add up to one. This is just a definitional thing at this stage. We define a convex combination as something like this.

If we had three points like this, we'll see that the set of all convex combinations actually looks something like this. If you do the math, you'll find out that as you vary ? 1 and ? 2 and ? 3, and you always have them nonnegative and you vary them so that they sum to one, the set of points that the sum traces out is called the convex hull of the set. We can take any set of points or just a set and we can form the convex hull by finding all of the convex combinations of points within that set.

Let's take these black points here. Supposed someone just handed you these black points and they said what is the convex hull of these points? At that stage, that sounds very unlikely, but later on, we'll see why you might be interested. We take these points and we want to know all of the convex combinations. This is the smallest convex set that contains all of those points. It's formed by basically taking the points at the edges of the set and adding the line segments involved in those points and try to make them as straight as possible. Then, you fill in the rest of the set.

So if you asked, for example, how I got to this point here, you could form it as a convex combination between these three points here. Or, in fact, this one lines just directly on a line between these two. For any set, we can find the convex hull of that set. Here's another example. The kidney shaped figure that we had before – to find the convex hull of this set, we need to just basically fill in this region here. That's the problematic region before. If we found two points like this, the line segment did not line entirely within the set. Any questions so far on convex combinations and convex hulls?

The main difference is on what you can do with ?. An affine combination of points that we had before is you have three variables. A linear combination – this would be a linear combination if we didn't have this restraint here. This thing makes it a convex combination instead of linear. Any other questions?

That's exactly right. If you already have a convex set and you want to know the convex hull, that's simply the convex set itself, because that set is required to contain all convex combinations.

The use of a convex hull – if you already have a convex set, the convex hull is useless to you. But sets are by no means likely to be convex in some sense, so it's useful when we want to make a convex set. But certainly if you call some function conf and you [inaudible] on a convex set, it would simply return the set without trouble. Yes, a convex hull has to be convex by that definition.

Let's have a look at convex cones now. Let's introduce a conic combination of points. This is again a specialized version of a linear combination of points. We take two points. We can have more than two. We have nonnegative weights on these points. That's described as a conic combination of these two points, X1 and X2. Here, we have a figure where we have a point X1 and a point X2, and this is the set of points ? 1 X1 plus ? 2 X2 where ? 1 and ? 2 are nonnegative.

Obviously, it's a cone, and that's why we chose the name convex cone. It also has to be convex. The reason for that is if we look back to our definition of convex, there was a special case of these. We added another constraint on top of this. If we merely wanted to insure convexity, we'd only have to check that X was in this cone when ? 1 and ? 2 added up to one. Any convex cone has to be a convex set.

Now we have a look at hyperplanes and halfspaces. Hopefully, you've seen these many times before. We start off with a definition. A hyperplane is a set where A transpose X is equal to some constant B. A is nonzero, because otherwise, we could have any point in the whole series. This is an example hyperplane. We parameterize it with a point, say, X, 0 and then we need to know that A transpose X0 is equal to B, and then any other point that also has an inner product with A of B is also lying on this hyperplane.

This is a hyperplane in two space, which is a line. In three space, you can think of it as looking something like this [inaudible] here. This would be a hyperplane in three space. It need not go through the origin, by the way. Following on from that, we have a halfspace, and this is similar to a hyperplane, but we just take the area on one side of that plane. We can have it either open or closed. This can be a strict inequality or a non-strict inequality.

This is an example here – same hyperplane, and it's the half space on the lower, I guess, in this case.

Likewise, the non-shaded region you could describe as another halfspace with A transpose X greater than or equal to B. The direction of this inequality indicates what side it falls on. A few things – halfspaces are not vector spaces. If you take two points from here, it should be closed. If it's a vector space, it would be closed under scale of multiplication. That's not going to be the case for halfspace, so a halfspace is not a vector space. People often get that confused, so don't fall into that trap.

That's a hyperplane and a halfspace. They are both convex. A hyperplane is also affine, which ensures that it's convex, but a halfspace is merely convex and it's not affine. It's not affine because we would have to be able to contain the line through any two points. Let's pick these two points here and then draw a line between them.

Student: Is there displacement from the origin and A and B that are written?

Instructor (Jacob Mattingley):There is, and it's basically if you find the point that's nearest the origin – I think this is part of the homework exercise. If you found that X1 was the point nearest the origin, then you could use A to find out – I'm going to leave that to you, because I think it is part of the homework. There is a relationship. Those are some very simple convex sets.

Some more sets that we're interested in – first of all, a Euclidean ball. At the moment, Euclidean is going to mean that we use the two norm in the ball description. We parameterize a ball by picking a center point XC and a radius for the ball of lowercase R. It's the set of any point that is [inaudible] in the Euclidean sense to XC then R. Here's one way of expressing that. Here's another way of expressing that. This is a free parameter representation where you have a ball U and it's just a unit ball. You scale it by R, and then you center it around XC.

This is the free parameter. Here are two [inaudible] descriptions of the same thing. We can get more interesting than a Euclidean ball and take an ellipsoid, which is a generalization of the Euclidean ball, and that's points – we take a symmetric positive definite matrix P and we use this expression here. You can actually go back and forth between this representation for an ellipsoid and this representation for an ellipsoid. This is just a sample ellipsoid. Ellipsoids and Euclidean balls are convex sets. One of the ways you could prove it is you could find two points and show that if those two points are in the set, then their convex combinations have to be in the set as well.

Again, if you haven't read chapter two and you haven't seen these things before, I'm going to be going quite fast. More generally, we're seeing two norms a lot. We're seeing the Euclidean norm. But more generally, we can define a norm as being any function that satisfies these things here. First of all, this is something like definiteness and non-negativity. We need to know that – these are just a bunch of things that you should look at that define a norm.

You can verify quite quickly that the Euclidean norm does in fact satisfy these, and there are various other ones as well. Just a little point is that this notation here is [inaudible] of the absolute symbol, which is a norm in the space R. If you take the absolute value of a number, it's going to satisfy all of these things in the space R.

In 364a, quite different from 263, if we simply have the norm symbol without any subscript, it does not necessarily mean the two norm, and in fact, it's usually used to represent a general norm. We've kind of moved up a level of sophistication, if you like, that no longer is it simply enough to just say this is the Euclidean norm of X, but we want to specifically say we are after the two norm. You'll find and use often different kinds of norms in this course.

We can define as we saw previously similar to the Euclidean ball here, for general norms, we can define a norm ball. This ball doesn't necessarily look like a ball. It's a ball in the general sense. We define them very much the same as we define a Euclidean ball, but they're going to look quite different. Let's take an example. We have the one norm, which is the sum of absolute values. Can anyone tell me what the norm ball with the one norm looks like in R2?

Student: Square.

Instructor (Jacob Mattingley):Okay. It looks similar to a square. It's actually a diamond like this if I recall correctly. The one that is a square, however, is the infinity norm. That's the maximum absolute value. That does indeed look something like this in R2. These are two other norm balls. They don't look much like balls, but that's their name. You could call this the one norm ball and this the infinity norm ball. There are others.

We can also define a norm cone by adding another variable here and saying the norm cone is any point X, T where the norm of X is less than T. This is our favorite norm, the Euclidean norm, and it's the norm contrast out. It's got a couple of names. It's called an ice cream cone for obvious reasons, and it's also called a [inaudible] cone as well. This is actually a solid figure. It might look like a wire frame, but it's actually filled in. It's any point where X is in R2 here along the bottom and T is up here.

One of the things we know about norm balls and norm cones is that they're convex. Yet again, we're adding another building block, which will become very useful to us later. It doesn't necessarily mean a whole lot now, but just store away that norm balls, norm cones, some of these other things like halfspaces – they're all convex, and that's going to be very useful to us later.

Before I go on, I'll mention that if you want to know what the one norm cone and the infinity norm cones look like, you can think of these here as a cross section when you look down from the top of the cone. If you looked down on this cone, the level sets of the cone would look something like this. If you looked down on the level sets of these cones here, the cross sections are going to look like this. We could fill this in and similarly for the infinity norm. You can think of them as a pyramid, and the one norm is rotated and the infinity norm is a [inaudible] pyramid.

Here, we have as lightly more interesting set of shapes that are also convex. A polyhedron is the solution set of finitely many linear inequalities and equalities. Here's one of the ways that we can represent it. We can say a polyhedron is any X such that AX is less than B and CS equals D. That would be a polyhedron. We note here that there's actually a slightly different symbol for less than here, and the reason for this, and you'll often see this both ways. You'll see an ordinary straight symbol here. You'll sometimes see this curvy one.

The reason is because this here is what we got from R. If we say three is less than or equal to five, this is what we use. Here, we're actually overloading this notation to say that AX, which is a vector, is component wise less than B, and to make this completely explicit, we're using this slightly different symbol here.

During this lecture and other times, I might get a little bit sloppy and the curves might not be as obvious, but the reason for this is because it's actually not quite the same as the ordinary R ordering, and we'll see a generalization of this quite soon. Note that it's different, and the reason is just to make it very clear that we're doing something different to what we were doing before in R.

Positive semidefinite we'll see in a second. This is one example of a polyhedron here. This is an intersection of a finite number of halfspaces and hyperplanes, and so in this particular case, we've got five halfspaces defined by these normal vectors A1 through A5, and you can think of this here as the halfspace defined by A1. If you find the intersection of these five, you'll discover that it's this shape here.

A polyhedron is convex, and this is something that you need to prove. I'm not going to go through it at the moment, but you should be able to show fairly easily that when you find an intersection of a finite number of halfspaces and hyperplanes you get a convex set, and it's called a polyhedron. An affine set is a special case of a polyhedron, and so is a hyperplane from before.

Student: This one only uses the inequality, so could you give a quick example of one that actually uses equality?

Instructor (Jacob Mattingley):Okay. Here's one. Pretend that this is actually an R3, and we've got five inequalities like this. We've got an equality which constrains us to being in this plane. They can be anything like that.

Next, we have a look at the positive semidefinite cone, and that's what you were asking about before. We say that a matrix is – you should know about positive semidefinite matrices by now, but we define a couple more sets. First of all, Sn is the set of symmetric [inaudible] matrices. Let's have a quick look and see. Do we suppose this is a convex set? Any suggestions? Yes, it's a convex set. How would we go about proving it?

Let’s use the definition. We’ll pick an X, which is a symmetric matrix. We’ll pick a Y, which is also a symmetric matrix. Our requirement for something to be convex – if this is a convex set, then if we take a ? which is greater than or equal to zero and also less than or equal to one, and we take ? X plus one minus ? Y, the set here is convex if each of these things here is also in the set. Is that true for a symmetric matrix if you take a sum of symmetric matrices? Is it still symmetric? Yes, it is. That means that this is a convex set.

It is, in fact, affine, yes. Again, you’re quite right. The reason for that is because we never used the fact that ? was between zero and one. ? could be anything and we would still have a convex set. Yeah, it’s all of these things. Let’s look at a more interesting set. We’ll define and use this notation extensively that S sub plus sup N is the set of positive semidefinite N by N matrices. We’ll assume and especially when we say is, we’re assuming that they’re symmetric.

This is a set like this. If you’ve forgotten what positive semidefinite means, it means that when we take a quadratic form such as Z transpose XZ is greater than or equal to zero for all Z. Here’s the claim, that S plus to the N is a convex cone. Looking back to convex cones, that means that if we take a nonnegative combination of two of these items from the set, it also needs to be in the set. You should be able to see quite easily that if we had X that was positive semidefinite and Y that was positive semidefinite, if we took a conic combination, ? 1 X plus ? 2 Y, this, too, is going to be positive semidefinite. Can someone tell me how I can make that really explicit quite quickly?

Student:Z transpose on the left and [inaudible] right use the distributive property to just –

Instructor (Jacob Mattingley):Exactly. I’ll do it like this. I’ll say Z transpose ? 1 X plus ? 2 YZ like this, and then I’ll expand it out and I’ll get ? 1 Z transpose XZ plus ? 2 Z transpose YZ. We know that this is positive or nonnegative because X is positive semidefinite. We know the same about this. We know that ? one and ? two are both nonnegative, and so the whole must be nonnegative. I’ve just shown that if we take a conic combination of two positive semidefinite matrices, the result is positive semidefinite. This means that S plus to the N is a convex cone and furthermore, it’s also convex.

We can do a similar thing for positive definite matrices, and here, we just add an extra plus to indicate that it’s strictly positive definite and not just positive semidefinite. Here’s an example of what the cone looks like in S2++. This is a cone of positive semidefinite matrices. You do write something like this. Exactly how you do it in MatLab, I’m not sure, but if you want, you could pick a point in the space – say we pick this point here.

You could find out its X, Y, and Z coordinates, plug it into the matrix and you see what the minimum [inaudible] value is. If it’s greater than or equal to zero, then it’s in this cone. If it’s less than or equal to zero, it’s – well, if it’s less than zero, it’s outside the cone. If you did a grid search in here, you could build up a set of points which are inside the cone.

Student: Can you have a positive definite matrix that's not symmetric?

Instructor (Jacob Mattingley):Yes, you can. This particular picture, it does depend on it because we've explicitly said that. Basically, it's bad form to give somebody a non symmetric matrix. This proof here nether used the fact that that was symmetric, so yes, it's true that the matrices don't have to be symmetric, but there's never really any reason why we would [inaudible]. There's no harm in taking any matrix said to be positive semidefinite and immediately symmetrizing it without loss of generality.

We could actually find an explicit form for the determinate of this matrix. It also needs the fact – what's that? Anyway, we can generate this plot. Feel free to look into how you do it. I'm going to move on for now because we do have a lot to get through.

Why did we use this particular sign here rather than an ordinary one? That's a good question. We have specifically used this curvy sign, and that is again because we're not using the ordinary ordering on R. This is now a matrix that we're saying is greater than or equal to zero in some sense, and it's the sense of positive semidefiniteness. We have again very deliberately chosen to use this different sign to avoid confusion with the pure, ordinary symbol like this. You will find some people who use that symbol, and that's okay.

So now we've looked at some sets that are by themselves convex sets, but we also want to know how to make convex sets out of other convex sets. There are certain operations that when we apply them to the set, they will preserve convexity. What it means is that you'll be able to build up more interesting sets from these simple ones that we started with. If we were constrained to one set, we could explicitly write equations and explicitly prove we're convex. We'd have a lot of work to do if we cannot with an unusual set.

First of all, let's have a look at some of these things. One of the ways we can see if a given set C is convex is we can just apply the definition. Sometimes, that'll be easy. Sometimes, it won't be easy. Another way we can do it is we can take operations that are known to preserve convexity. We can take these little [inaudible] that we know are convex, and we can put them together and quite quickly make a fairly interesting looking set. Here's one. It's an intersection.

This is actually any number of intersections, and I think we look at that on the next slide. An affine function will preserve convexity. If we take a set C, put it through a function F which is affine, it's going to preserve convexity. A perspective function and linear-fractional – we'll see those in just a few moments.

Here's another way. This is the crude and nasty MatLab approach. Say I want to know if a set is convex, and say I've got some oracle which tells me whether a given point is in the set or not in the set. Here's what I can do. I can pick a couple of points, say, X1 and X2 completely at random from the space that I'm in at the time. Then, I can test my oracle and I can see if X1 and X2 are in the convex set. Suppose they are. The requirement of convexity is that when I find this convex combination of these two points, that point itself has to lie in the convex set.

If somebody gives you some random set and you just want a very first pass of is this even remotely likely to be convex, here's what you do. You make a tiny little window in the corner of your screen. You run MatLab and you start generating random points from the set. You take a convex combination and you test to see if it's still in the set. Now, as soon as you find some ? which is between zero and one where ? X1 plus 1 minus ? X2 is not in the set and X1 and X2 are, you immediately return and say it's obvious. Of course it's not convex. You can see that when you're looking at it. It's quite easy to do that.

Here's another thing, which I won't go into at length. Basically, midpoint convexity where you just take ? equal to 0.5. That's actually enough for some technical conditions. If you want to run a whole bunch of tests, you could just find X1 and X2 and take their average. You test to see if the average is in the set. If it isn't, you return and say no way it's convex. If it is, you don't necessarily know much. It's only useful if it's not a convex set.

But if you did, say, 100,000 points, you could give a very rough first pass. Note the quotation marks – it's very rough. It's a heuristic method. Nevertheless, it can be a very fast way of finding out whether something is convex. It fails in situations where – it's to do with boundary issues. I'm not going to talk about these much, and the focus of this class is not really on the technicality, but if we took these points here – in this particular case, it would work. Say that point there was in the set. Then these other points wouldn't be anything [inaudible].

Let's have a look in more detail at intersections. Here's the claim – the intersection of any number of convex sets is convex. Fortunately, I can just refer you to your homework, because I think you have to show this in your homework for a couple of sets. When I say any number, that means finite, infinite and unaccountably infinite. This is a very robust way of finding out whether something's convex.

Here's an example. Here, it's an unaccountably infinite set, I think. We just take this random thing here and we know that the individual elements are convex. We put them all together and we put up with this shape here. I don't want to dwell on the shape. It's colored in lots of detail in the book, and in fact, let me just take a moment and mention the book.

The convex optimization book is available, and these slides are based very much on the book. Everything here is covered in detail in the book. The book is also correct from a mathematical point of view. If I'm tossing around boundary issues and there are little technical differences, we're not going to worry about them in lecture nearly so much. The book is correct. If you have done analysis and you are concerned about all kinds of things like closure and compactness, then you can go to the book and it will be correct. We're not going to worry so much about that now.

Here’s a set where we have an infinite intersection of convex sets, and it is itself convex. Let’s have another look at a way that we can preserve convexity. Let’s have a look at affine functions. Suppose we have some function F that takes RN into RM, and it’s affine. One way of looking at this is saying that if F of X is AX plus B, A is a matrix and B is a vector. This is actually not the most general way of expressing it, but this is pretty good.

The image of a convex set under the affine mapping F is convex. If you’re given some set S which is just a set at this point and it’s RN, we can use the mapping F and all of the points from X that are mapped into FX – the new set is a convex set, and likewise for the inverse image. Just to make clear what the difference between these is – suppose we have some set S here, and this is a convex set. We take this under a mapping F to some other set here, this is the set F of S. This is a convex set if F is an affine mapping.

Likewise, we can do the same for an inverse mapping. If we had another set – these are not very interesting looking sets, but it could be any set. We can take this back via F and F is a convex set. After this mapping, it’s also a convex set. If we have an affine function, we can actually create quite a lot of damage. When we have a known convex set, we can make other ones out of it.

Student: Which notion of adversity – does it have to be one to one?

Instructor (Jacob Mattingley):Actually, we don’t really run into those issues. This is the definition here. It’s simply the set of any points where the function value of that point matches to the set C. It’s the inverse image of the set. The function – it is affine, so is it one to one? It probably is. It doesn’t have to be. It’s the inverse image, which is a well-defined thing. It’s always convex if the set C is convex.

Some examples of this – we have simple scaling by two. We have translation where we move around, and we have prediction. Let’s briefly mention that the converse is not true. We might have a set F of C that’s convex when C wasn’t convex. We’re more interested in this, anyway. Here are some simple examples.

Prediction could be really simple. We might have a set X1, X2. This might be a point in a convex set C, and if this is convex, then the set of X1 given that the sum – the sum of X1 – for sum X2, the set of just X1s might be a convex set. That’s a very simple prediction. A more interesting one is what’s called a linear matrix inequality or an LMI. This is an affine mapping. If we have a set of matrices AI and they’re all symmetric matrices, then the set X where this weighted sum of symmetric matrices and the positive semidefinite sense is less than B, that is also a convex set.

I’ll talk about that one for a second. If we take the cone – this function F of X – we’ll make it equal to B minus A of X where A of X is equal to X1, A1 up to XM, AM. This is an affine function, and it needs a little bit of hammering to get it in the same form as we saw above here. It is an affine function.

We already know that the positive semidefinite cone – we already know that this is a convex set. If we take the inverse image of the set here with this function F, what we get is that the inverse image is equal to any X and RN such that B minus AX is in the positive semidefinite cone. This is just a fancy way of saying that B minus A of X is less than or equal to zero in the positive semidefinite sense.

We've taken a very innocent looking set, S plus of N. We talked about how that was a convex set earlier. We took also a fairly innocent looking affine function, F of X, and immediately, we've got this fairly unusual looking LMI, which is actually a set of weighted symmetric matrices. That's what's on the left-hand side here. This is a convex set, and there were just two very simple ingredients. You can see that we can cause a lot of damage.

This comes up a lot in control, and it's a very useful set. Another one here is the hyperbolic cone. This one you can get in a similar way. We already saw that the set ZU, which is a cone when we restrict the U. We saw this set before. This is a norm cone, and we showed that this was a convex set. We know this is convex, and we want to get to something more interesting. You can see that if we apply the affine function F of X equal to P one half X and C transpose X, the inverse image of this mapping is also a convex set.

If you take this cone here and you map it through here, you get a hyperbolic cone. Again, that's something that will be interesting. You should verify the details and as always, details in the book if you need them as well. Here's some quite bizarre sets that are definitely not obviously convex, and you can show quite simply how they are convex if you know how to do it.

Departing even more from obvious sets, we've got a perspective function. This one's a little bit unusual to start with. If I have some function P that reduces RN plus 1 to RN by taking the last component and dividing – you can think about it as taking some vector like this and we pick off the last component. We divide each of these elements by that component and then we throw this away. It's a somewhat unusual operation, but [inaudible] one, two, three, I'd pick the three, and my new vector would be one third two thirds. That's a perspective function.

The images and the inverse images of convex sets under the perspective function are convex. That also is going to be useful to us later. Remember, we're just building up a box of tricks that we're going to find useful later. If it's a bit dry, bear with us for the moment. There's a demand constraint here that this last entry has to be positive.

A generalization of the perspective function is when we have a linear-fractional function. If we have some matrix A, a vector C and vectors B and D, then again, under both the image and the inverse images of a convex set under this transformation are themselves convex. There is a domain constraint on positivity of the denominator here as well. Just to see one example of this because it's probably quite a mystifying concept if you haven't seen these before, we take just a set C here like this.

Here's a linear-fractional function, which you can think of as similar to a perspective function or a generalized perspective function. If we hold this figure on the side – if I hold it like this and I look at this figure from a different angle, I'll find that these parts here are closer to my eye. If I look at this, I'll find that these things loom closer in my vision than the other parts of the figure. This is quite a strange transformation, and C here is obviously not convex. In fact, why is C not convex?

The line segment is very much not in C. If this were a convex set, though, under this transformation, it would remain convex. The reason it's not is because you wouldn't be able to see it too well if it wasn't convex. I'm actually not exactly sure what the dotted line is about. Actually, probably what it is – it's probably a line under the same transformation. I suspect that's what it is.

Another thing you can say about functions is if they transform line segments into line segments, it actually is a convex function. I suspect that's what it is, actually. You'd have to check me, but I think this line moves to this line, and because this line segment moves to this line segment, you can also say that it's a convex function. There are some slightly weird functions. Have a look at them later and put them away as things that we're going to find really useful to prove that sets are convex.

Let's have a look at generalized inequalities. Before, we saw moving from a simple inequality in R to a component wise inequality. It turns out that it's quite useful to consider other kinds of inequalities as well. Let's introduce proper cones first. A proper cone is a convex cone with another bunch of constraints.

First of all, the cone has to be closed. I'll just mention that some of you will have an analysis background where you know exactly what closed means and you have read many things about closedness. Some of you don't, and that's fine for the purposes of this course. The five-second introduction to what closed means is if we have a shape and the set includes the border of that shape, then it's a closed set. If it doesn't include the boundary at every point, then it's not a closed set.

If you know what closed means, that was probably an insult to you, but that's actually probably enough. It's an intuitive idea of what closed means. It's probably enough for this course. That's the first requirement to make a convex cone into a proper cone. We also say that it's solid. It actually has a nonempty interior. Again, interior is not a concept that some of you will know precisely what it means. Others of you probably haven't heard it before. You can think of interior from an intuitive point of view as the inside of the set. The set not including its boundary.

Tell me about this here. If I just have this ray here, is that a proper cone? Why not? This has an empty interior even though it's getting thicker by the second, but if I just have a ray out from the origin, it has no interior, so it's not a proper cone. There's one more requirement, and that is that the cone cannot contain a line. That means that this set here – this is a cone, and it's also a convex cone, but it's not a proper cone because it contains this line here through the origin.

These are the requirements. It has to be convex, closed, solid and pointed, and that gives us a proper cone. We'll find that that's pretty useful. If we didn't have these constraints, when we go to generalized inequalities, it'd fall to pieces quite quickly. You're saying that any convex cone includes a line. So what about this cone here? This is a convex cone, and it does not include a line. It includes a ray out from the origin, but it doesn't include a line through the origin.

Here are some examples of proper cones. First of all, the one you're most familiar with is the nonnegative orthant. If I have R2, for example, the nonnegative orthant is this portion of R2. That's a proper cone. It's convex. It's closed because it includes these boundaries. It's solid because there's certainly plenty of stuff in the interior. It's pointed because it contains no line, and so it's a proper cone.

Here's another example. The positive semidefinite cone is [inaudible]. That's probably slightly more tricky to prove, but all of these concepts generalize into this cone here. They're not a problem. Here's another one, the set of nonnegative polynomials. We're going to find that this is extremely useful and used all the time. We're going to use this a lot. This one is a more interesting example, but we probably won't see it all that often. Still, it's a proper cone.

This one here? It's a cone because it contains – the definition of a cone was just that if I have a point X in the cone, ? X for ? positive has to be in the cone. I put any point in here, and I scale it with a linear vector, and it's still in the cone. I pick this point here. I can scale it positively and it's still in the cone. I take this point here. I scale it positively. I stay in this area. You can't find me a point which won't scale and remain in the cone.

This is not a convex cone. This isn't convex, and so neither is it a proper cone. Let's have a look now at generalized inequalities. Before, we had this symbol with the curvy lines. Now we add a K to it. If we take a proper cone K, we can parameterize a generalized inequality with this proper cone K. We describe it as being less than Y. If Y minus X is inside this cone K, we can [inaudible] strict inequalities in a similar way by saying that X is strictly this if Y minus X lies in the interior of the cone K.

Let's have a look at a quick example. The component wise inequality that we're familiar with – this says that X is less than Y in the component wise sense if each of the components XI is less than or equal to YI. Let's take a couple of points – this one here and this one here. Let's call this X and this Y. X is described as being less than or equal to Y in this proper cone K if Y minus X lies in the cone. Let's find Y minus X, and it's this thing here. Y minus X is not – X minus Y. Is this backwards?

Okay. Y minus X – what's going on here? Y minus X is inside here. If I had a vector 3,3 and a vector 1,1. Subtract them and I get 2,2. It points to Y, and this is inside this proper cone here. That's a component wise inequality. We can say the same about a matrix inequality. We've seen these before. X less than Y is this PSD cone here means that Y minus X is positive semidefinite. These are so common that we drop the subscript. Not only that, but we often also drop the special symbol and just use an ordinary symbol like this.

For the purposes of clarity, it’s a good idea to stick with this. If it’s obvious what you’re talking about, you can drop the K and sometimes people even just use an ordinary symbol. As you’d expect when we’re using the same symbol generalized, a lot of the things that we saw before carry over, and the properties are analogous. We have things like this here, and this property will carry over. Be careful, because not everything does.

One of the things that doesn’t carry over is we don’t have a linear or a total ordering. That means that if we find X and Y, X may be neither less than Y or greater than Y. Here’s another way of expressing that. This generalized inequality here is not in general a linear ordering, so like I just said, we may have neither of these holding. You can see that quite easily in our usual case here. If I have a point here and a point here, then this point in the sense of component wise is neither less than this point nor is it more than this point. It’s not a total ordering.

We now introduce another concept in this lecture, that X in a set is described as being the minimum element with respect to this generalized inequality if any other point inside the set is actually more than X in this case. That’s one concept, and we’ll have a look at a picture in a second. A second concept is a minimal element. We describe an element X as being minimal if any point that is less is actually the same point. That’s a little bit bizarre to start with. Let’s have a look at that.

We’re going to use the cone K is equal to R plus to the 2, so component wise inequality in R2. This point X1 here is the minimum element of S1. What that means is if I picked any other point Y in S, then I can explicitly say that X1 is less than or equal to Y. That will hold for any point in here. Let’s have a look at the difference here in S2. What we need here is that I picked X2 and S2. X2 is a minimal element. That means that if I pick some other point here, I know that if Y is less than or equal to X2, then it means that Y is equal to X2.

Here’s a point, for example. Make that Y. X2 is not the minimum element, because for this way, I cannot write X2 less than [inaudible] to Y. I can’t actually do that because it’s not a linear ordering. That’s not allowed. But I can say that any point that is less is actually the same point. That’s because – here’s another way of thinking about it. If I take this cone K, this set here is the set X1 plus K, and that’s the set of all points that are unambiguously more than X1. They can always be described in reference to the cone K as being more than X1. X1 is unambiguously least.

In this set, if I drew the same figure here, these are all of the points that are unambiguously more than X2. This point here is not – this point Y is not in the set, so I can’t say that X2 is unambiguously less than Y. What I can say is I can form this other set X2 minus K, and this is the set of points that are unambiguously less than X2. Any point in here, I can say with full authority – this is a point that’s less than X2. That means because I find that there are no points inside the set S2 that are unambiguously less, I can describe X2 as being a minimal element.

It's a slightly confusing terminology here, but here's how it goes. A minimum is something that you cannot under any circumstances claim that there's a least point. All points are more is what you say for minimum. It's very confusing, even when you've seen it before. Minimum – all points are more. Minimal – no points are less in this unambiguous sense.

We could have a different cone. We could have this cone here. This is a proper cone. We could call this K2. Then, we could again look at this and we could say – we could talk about this relative to K2. How would you describe X1 relative to the generalized inequality parameterized by the cone K2? How would you describe X1 in terms of this one? It's minimal, so it's no longer minimum because the set doesn't lie inside this cone. There are points that are incomparable, if you like. There are points that you could argue are actually no more than this in an unambiguous sense.

That's going to be useful to us. Let's introduce another concept while we're on a roll. This one is the separating hyperplane theorem, and it says that if we take two general convex sets, C and D, and if they're disjoint – they have no intersection and C intersect D is the empty set. If C and D are disjoint, then there is some A, not zero, and there's a B so that the halfspace A transpose X in C lies in a halfspace A transpose X less than or equal to B. The opposite holds for X and D.

Here's a couple of sets. Here is a hyperplane between the sets, and if F is in R2, then it's a line. You can also think about it as looking down – a prediction of R3. This says that we can find some hyperplane so that D lies entirely in this halfspace here, and C, if you like, lies entirely in this halfspace here. It's called a separating hyperplane for quite obvious reasons. This theorem, which I think is proven in the book, says that any two sets like this that are disjoint – there has to be a separating hyperplane.

Strict separation requires more, so I won't go into the details here, but strict separation would be the hyperplane that actually passes through neither set. We can use this separating hyperplane theorem and turn it into another one called the supporting hyperplane theorem. A supporting hyperplane we define as being one where – this is described as being a supporting hyperplane if the only point that it touches is – if it goes through X0 and all of the set is on one side. You can say that if C is a convex set, then at every boundary point of C, there will be a supporting hyperplane.

Let's try to connect these two new ideas together. We have a separating hyperplane theorem which says if we have two sets and they're disjoint, we can find the hyperplane that separates them, and one which says that if we have a convex set and a boundary point, we can find a supporting hyperplane. We can actually use the separating hyperplane theorem to prove this one. We can actually say – we'll form a set X0, and we'll also form a set interior of C.

We take a convex set C. We'll assume that it's convex by looking at this part of it. We'll say that X0 is one set and it's convex because it's just a single point. The interior of C is a convex set if C is convex. The interior of C and X0 do not intersect if X0 is on the boundary, and then we can find a separating hyperplane between X0 and the interior of the set. That will be a supporting hyperplane.

How did I use the convexity? It's actually used in the proof. This is a non-convex set, and if I had a point here, I cannot find a supporting hyperplane, because any line through here is actually going to cut through the set. The theorem is that if C is a convex set, then at any boundary point, I can find a supporting hyperplane. The other thing I used is that the interior of C is convex if zero is convex, and so C and D are disjoint convex sets. I used convexity of those sets and the fact that they're disjoint to show that there is a separating hyperplane and I used that to show that a supporting hyperplane exists.

The square we saw before – its interior was convex, but nothing special. This holds for any convex set. If the set was the interior of C, all of these theorems would hold. It's just a convex set. In fact, if we look at this one here, I could have a hyperplane though here. This also is a separating hyperplane. Here's a convex shape, and this is a supporting hyperplane. So's this. These are not unique supporting hyperplanes and separating hyperplanes.

Here's another one. We are powering through the material, and again, I encourage you to have another look through chapter two. What we can do now is introduce a dual cone. Dual cones again are somewhat strange when you first see them, but they're going to be very useful to us later. You'll be seeing the world dual, duality and other things like that every week throughout the course. This is defined for any cone K – it doesn't have to be convex. It surely doesn't have to be proper. We define the dual cone, K*, as equal to any point Y with inner product greater than or equal to zero with a point X in the cone K.

Let's have a look and see what that means. If I have a cone that looks something like this – I'll call this K. It happens to be proper, but that's fine. K* in this case is any vector Y where Y transpose X for all X and K is nonnegative. Let's take this line here. Draw out a right angle here. That's a right angle right there. I take this one here and I do a similar thing. This set here is actually the dual cone of K. This one here is K*. That's the set of – if I take any vector in here, its angle to all elements in the set here is less than or equal to 90 degrees.

If I look at this vector here or any vector in this dual cone K*, its angle to all elements of K is less than or equal to 90 degrees. This is the dual cone. A few examples – if I look at the nonnegative orthant from before – that's the set here – the dual cone of the nonnegative orthant is the nonnegative orthant itself. It's described as being self-dual. You can see that quite easily if I have two nonnegative vectors Y and X, Y transpose X has to be less than or equal to zero, and you should be able to go through a proof of that quite easily. We could just do YI, XI and so if the components are nonnegative, so is the product and so is the sum.

The positive semidefinite cone – that's also self-dual, so in this particular case, we actually use a slightly different definition of inner product, and again, details are in the book. If you find the set here, it's going to be the set itself. Euclidean norm cone – it's self dual, but finally, we get to one that's not self dual, and in fact, most of them aren't. A lot of the useful ones are, but here's one that isn't. It's the one norm cone. The dual cone of a one norm cone is actually the infinity norm cone.

The other thing that we note is that the dual cone of a dual cone is actually the cone – the initial cone that we were after – if it's a proper cone. That's with a big if, if K is proper. It may also hold if K is simply convex. You'd have to check me on that. What we can say now is that the dual cones of proper cones are proper, and hence, we can define generalize inequalities. If you give me some inequality with K, I can define a generalized inequality with K*.

We talked about minimum and minimal elements before, and here is one of the ways that those come into play. We have some set S. We know nothing about the set. Let's say that X is the minimum element of S. That is true with respect to the generalized inequality. X is the minimum if and only if for all ? that are strictly in the dual cone if X is the unique minimizer of ? transpose Z over S, then X is the minimum element.

This is just touching on the very first parts of duality, which is going to come in lots in this course. It's a little mysterious at first, perhaps, but here's a definition. It's going to be quite fascinating what the ? means exactly. For now, that's what a minimum element is. A minimal element is slightly different.

If X minimizes ? transpose Z over S and there is some ? – if we can find any ? that's in the dual cone, then X is minimal. We'll see later how useful this is. Finally, we can say that if X is minimal, there is some ? such that ? transpose Z has a minimizer X. These things – I'm really starting to run out of time, so I'm just tossing them out there, and we'll see duality in lots more detail.

Last thing today is what we call the optimal production frontier. You've probably seen this before, especially in 263. We talk about it with a very simple production frontier. Here's what it says. P is a set of possible production methods. On this axis, we have fuel, and on this axis, we have labor. Say we're testing out different production methods in our factor. If a point is in the set P, it says that there is some production method, which, say, produces 100 milk bottles and takes this amount of labor and this amount of fuel.

If X is in P, there is some possible production method that uses that amount of labor and this amount of fuel. This is a very simple example in R2. This set we could find. Some marketing guy might tell us what products are available. We might do some research. We might do some testing. There are lots of different ways we could find this set. What we say is that an efficient method is one that is minimal with respect to R plus to the N.

First of all, what would it mean if we had a minimum element? Let's say we had X and P, and we're using this generalized inequality with R plus to the N. If X was a minimum in the set, it would mean that if I had X here, say this was in the set. The whole of P would lie within this. It would say that any production method requires more labor, more fuel or both than X. That would be X being minimal. Minimum is a slightly different quality.

If I had X1 here, this is a minimum element. What it means is that there is no method that requires simultaneously less fuel and less labor than X1. It's efficient. Any vector along here – there is no method that is unambiguously better. There's none that simultaneous require less of both. Labor might be more expensive than fuel, so we might want to move along the curve and trade off between them, but we don't want to move off this curve. Likewise, this part of the curve is also efficient methods.

The problem with X5 is we can produce at X5, but we could produce at X2 and we would use less labor and simultaneously less fuel. There'd be no point in picking the method X5 if these were all the parameters involved. X1, X2, and X3 are efficient, and X4 and X5 are not. You can think about these as being minimal with respect to R plus to the N. There are other ways. We might be interested in a different cone, and there might, in fact, be a minimum for a different set, but this is just tying at least one of these things into a practical application.

Any final questions? We are done, and Professor Boyd will be back on Tuesday.

[End of Audio]

Duration: 77 minutes

ConvexOptimizationI-Lecture03

Instructor (Stephen Boyd):All right. Now I guess I'll take that as the – as a sign that we're on. So actually, first of all, embarrassingly, I guess I have to introduce myself even though it's the second week. And I have to apologize for not being here last week. I've – actually, for the record, I've never done that before.

I've never missed an entire first week, and I promise I won't miss any more classes, maybe, mostly, it's – anyway, I'll make a very – if I miss anymore classes, it'll be for a really, really good reason.

Anyway, you were in very good hands with Jacob, so that's okay. A couple of announcements [inaudible], I'll start with a bunch of administrative things. The first is that we have two more TA's. This is Argerio Zimnus, who is sleeping, as most senior PhD students would be at this hour, and Cwong Mo Co is over here.

What we'll be doing is adding more office hours both for them. We now have – since we now have essentially like an army of TA's, we'll – we can have – we were joking about just having office hours like 24/7. I don't think we can pull that off yet, but we're going to do something like that, and in fact, I do have some questions. Also I'm going to add office hours for myself, and we don't know – we're trying to figure out what – I don't know, if people have opinions about where we should distribute the office hours, so I don't know. We have our own conjectures, but if anyone has any other?

For example, in my case I might add them Tuesday and/or Thursday afternoons. What do you think? Yeah? Oh, that was a yes, oh, okay. Okay, all right, fine. Then I'll do that. I'll add office hours for myself, and for the TA's.

Now this week, of course, is chaos week, otherwise known as the equals' week. If you're not in this department you'll – you actually – you cannot even imagine what this week is like, but just look around you and you'll get the idea.

If you're in this department, you know full well. So those office hours, at least in my case will kick in – have to kick in next week, but look to the website for that. Let's see, a couple of more administrative things. Here's one, and I – it has to do with the homework.

What we're going to do is we actually did a little bit of planning for the homework, and the way it's going to work, and actually it's a good excuse for me to say a little bit about how the course is gonna go for a while, and this is actually substantive.

Homework 1 and Homework 2, Homework 1, which you've already started – yes? Okay, so Homework 1 and Homework 2 are going to be just these big long slogs through – I don't know, it's – you'll have nightmares about it, and people coming up to you on the street and saying, is this set convex, is this function convex, and things like that for years.

But the nightmares will subside eventually. They'll always be there to some extent, so there'll be little triggers later in your life that will set it off again. Someone will say something like, which of the following, and you'll – something will remind you of it and you'll get all on edge.

But seriously, what we're going to do is this; we're trying to go fast over the first material, which is actually a little bit, I mean, it is interesting. We're going to use all of it eventually, but it's sorta the mathematical basis of what we're gonna do, and so the idea is to actually not cover it as well as we should.

Meaning, we're going to go fast. Faster than you – there's no way you could kind of absorb every subtlety. So what we'd like you to do is read the book. I mean, that goes without saying. Read the book, you'll do the homework, listen to the lectures, and you'll get maybe 75 percent of it, something like that.

Actually, that's enough because what's going to happen then is in – by Homework 3, it'll actually get interesting and we'll be doing a lot of applications from then on, kinda mixed in with new material, and at that point it gets interesting. So the idea is to avoid having something which for four weeks, non-stop, is essentially just a math course, so that's sorta the rational behind this.

On your part what it means is this; if you think we're going fast, you're right, we are. But it also means don't worry about getting absolutely everything. All the things we're gonna to see you're gonna get lots more chances, and it'll be more interesting later because it's gonna be in an applied context.

Having said that though, this is your time to kinda get down some of the basics. Okay, so that's our plan for how the course is gonna go. I should also say that there was one change. I don't know if it was announced yet, and that is that the homework we're gonna try to have graded a little more seriously.

So the – then for example 263; in 263, the homework's were graded on a scale of 0 to 4, very crudely. That's nominally two bits. The actual amount of information from the entropy was on the – of .2 bits. We haven't worked out actually what the entropy is, but it was something like that.

So we're going to try to do – at least give you a little more feedback and make this 0 to 10. It's going to be imperfect, but we're going to attempt that. So it's going to look something like that. The idea is if it's perfect, that's a 10, if you – anyway, we'll – when we – we'll get into that later.

Another admin – maybe just two more admin things and then we'll move on to real material. The other is that we posted another reading assignment, and so right now you should be reading Chapter Two, and when you finish Chapter Two, keep reading, just go right on to Chapter Three, and four for that matter after that.

And then let's see, one more admin announcement. This is a bit specific, but it's a good excuse to mention something. We got several questions about what do inf and sup mean, and that's a good excuse for me to say a little something.

So, you are welcome – if you don't know what these mean, you're welcome to treat these as min and max, okay? So, two – one thing is actually for questions like that. When we get more than 10 or 15 questions, we're going to start a FAQ, on the website, so please let – what? Unknown Speaker:

It's already there.

Instructor (Stephen Boyd):Sorry, it's already there. Okay, it wasn't there two minutes ago, but anyway. So there's a – there's going to be a FAQ section in the website. Already pushed over? Okay. So there is a FAQ section in the website. And so please check there if you have some sort of basic question first. And this is anything ranging from, I can't get this software to work, to what do inf and sup mean, or when is Homework 3 due, or I don't – I guess that's posted on the website, but that kind of stuff. We'll just post stuff there.

And if you have suggestions, actually, for us to add stuff there, just send us an email. Okay. And on this topic, I should say a little bit; if you have had analysis, of course on mathematical analysis, then you know what infimum and supremum mean. If you haven't, I would simply substitute min and max.

And the way the course is going to work is something like this; the book, by the way, is not – I mean, there are many more mathematical books, so if that's your interest on your website as pointers to other books, you can go insane. And it can be as formal as you like.

By the way, outside the lectures you're happy to hunt me down. I'll tell you what I know, if this is your interest. Frankly that's not really the whole point of this class, but I'm happy to answer any question about that.

And I'm talking about things like detailed conditions, is this open, is this closed, what happens if the boundary – all these kinds of things. You know, what are the exact conditions under which this holds and things like that. In lecture I'll be very caviler about these things.

So boundary conditions don't hold, everything I say is modulo, some technical conditions. In the book we, generally speaking, don't lie. If you look in the book there's just a few things that are wrong. Well, you can treat it as true, the book. But in lecture I'll just be happy to just go broad brush over these things.

And I think the class, by the way, is – just from past experience, if you've had – if you have a deeper understanding of all the analysis and stuff like that, great. But I haven't noticed that it makes any difference in the experience. You don't have to know it; you can work with an intuitive idea of these things. Not really worry too much about boundary issues and things like that and you'll be just fine and totally effective when – at using these methods.

Okay, so that's all my admin stuff, any other questions? Otherwise, we'll jump in. So today we'll cover convex functions. And let me say a little bit about how this is going to work. If you go down to the pad here, that would be great. There we go.

So we're going to look at convex functions. First of all, just define what one is, and then look at various aspects of it. And let me say operationally why you need to know this; because one of the things you're going to need to do is actually determine if a function is convex because to use this material, that's like the most basic skill.

So that's why we want to – that's why you want to look at all this and see how this works. Okay. So let's just jump in. Well a function is convex if its domain is convex. That's the first requirement. And the second is that it satisfies this inequality for theta between zero and one.

So this says that if you evaluate F at – this point you saw in the convex sets lecture. This is a convex mixture, a convex combination of X and Y. So geometrically it's a point on the line segment between X and Y.

This says if you evaluate a function at a point on a line segment between X and Y, the result is actually less than the same mixture of the values of the end points. Or in terms of the graph, it says that if you take two points on the graph of the function and then draw the straight line that connects them, and I guess an old word for that is the cord, or something like this. Right, so that's a very old word, is the cord, and it says – this says the cord lies above the graph.

So that's what – that's what this inequality says. And actually you'll get a very good idea of what this means eventually. Another way to say it just very roughly is upward curvature. So it just means curves up, that's all. And by the way, of course, a convex function can look like that. It can be monotonic decreasing. Nevertheless, the curvature is upward for a function like that.

It doesn't have to be bowl shaped, but it should have positive curvature. Okay, now you say a function in concave is –F is convex. That means negative curvature, downward curvature, something like that. And it's strictly convex if it's convex. And not only that, but this inequality here holds with strict inequality provided data is strictly between zero and one. So that's strict convexity. And you have strict concavity too.

So let's look at some examples; well – and these are just on R. So these are just functions you can draw. So the first is just an affine function, so that's linear plus a constant. That's – it has zero curvature, so it's convex. And in fact what happens for a function that is affine is the following; is that in effect you have equality here. For always, for an affine function that's exactly what it means.

It means that if you do a linear interpolation between two points, you actually get the exact function value. So that's essentially the boundary of a set of convex functions. Exponential, doesn't matter what the coefficient is in here, this is convex. So if A is positive, it's increasing, but the curvature's upwards. If it's negative, it's a decreasing function, but it's convex.

Powers separate out. It depends on the values of the exponent. If the exponent's one or bigger, or if it's negative, or – well zero [inaudible] that's just a constant one, in that case it's convex. I mean, these are things you can just draw and see. So I think the question is to whether or not a function on R is convex, is a non-issue.

Here's how you check; you draw it, and you use your eyeball to see if it curves up. So there's really no issue there, okay? So these – we'll find formal ways to verify all these, but I think in terms of a function on R, there is no issue. You just – you draw, does it curve upward or not, that's the question.

And you have things like power of absolute value would be another one. Negative entropy is X log X. That's something that goes like this; it's got this infinite slope here and then goes like that. Something like that, and – but at point as it curves upwards, that's negative entropy.

So entropy, which is minus X log X is going to be concave. Okay? Now in concave functions examples would be like an affine function, and in fact, of course, if a function is both convex and concave, then it's affine. And that's clear because it says basically; this inequality holds one way and the other. That means its equality. This in fact implies the function is affine.

Okay? This power's in the range between zero and one, you know, like square root for example, you just draw this, and it's clearly concave. Log rhythm, it's another famous example. Okay. A little more interesting example on RN and RM by N, that's the set of M by N real matrices.

So here's – of course you have an affine function on RN, that's [inaudible] plus X plus B. That's a general linear function plus a constant. So this is the form of a general linear function, affine function on RN. And that's going to be convex, it's also concave.

Norms; so any norm is convex. That follows actually from triangle and equality, or – I mean, that's part of the definition of being a norm. And examples are things like this; the so-called p-norms, which is the sum of the absolute value of XI and then to the one over P. Now for P equals one, that's the sum of the absolute values for P equals two. It's usually Euclidean norm, but for example, for P equals three, it's the three norm, which is the cube root of the sum of the cubes of the absolute values of a vector. Yes?

Student: Is there a half norm?

Instructor (Stephen Boyd):That's a very good question, is there a half norm? So some people would – let me answer it first – well let me just first give the answer. The answer is no because the square of the sum of the square roots is actually not a convex function, and therefore it can't be a norm because norms have to be – all norms are convex. It's actually concave.

Now having said that, it is currently very popular in statistics and a bunch of other areas to use a – as a eristic for finding a sparse solution that involves – we'll see this later, by the way, using things like the so called p-norm for P less than one, but it's not a norm, so it's weird.

Student: Why do all the norms have to be convex?

Instructor (Stephen Boyd):That actually follows from the definitions of – norm has to satisfy a triangle in equality and a homogeneity property. And then from those two you can establish it has to be convex. By the way, I should mention something here. It's not easy to show – if you put – just for general P, just take the three norm. It's not easy to show that's a convex function. It's not easy at all. So it's – I mean, it's not hard, but you have to know exactly just the right five or six steps to do it.

So this is maybe the first thing I've said that wasn't trivial, and it's not. Okay, let's look on matrices; actually, every now and then we're going to – there are going to be problems where the variables are matrices. Sometimes square, sometimes they'll be rectangular, but let's look at a couple right now.

What is an affine function on matrices? Well the general form looks like this. A trace of A transpose X plus B – by the way, when you see this you should read this as follows; this – by the way, some people write this as the inner product of A and X plus B. That's the standard inner product on matrices, is trace A transpose X.

If you work what it is entry by entry, it's just this. So it's as if – well it's this, let me write it another way, let's see if I can do this. If – that's not totally – that's mixing weird and – that mixing notation, but the point is this says that if you string out A as a vector, string out X as a vector and then calculate the ordinary inner product, well you would just get this, like that. Okay?

And that's the same as this thing. So this notation, you might as well get used it. When you see trace A transpose B – A transpose X, that really means you can think of it this way. And, by the way, another notation for this is A big dot X or something like that, so you'll see that.

Okay. Here's an interesting function, which is the norm of a matrix. So that's – and I'm talking about here, the spectral norm or the maximum singular value, or you can call it many other things, and there's not that many other – actually there's probably just a couple other names for it, the L2 induced norm and maybe the – now I'm running out of obvious ones.

So this is the square root of the largest [inaudible] X transpose X. Now I want to point something out, that is very – that's a very complicated function of a matrix. So that's – it is not a simple – to take the matrix form X transpose X, find the largest [inaudible] value of it then take the square root of that.

That's a chain of quite complicated operations. So that's a function which is not simple, but it's a norm and it's convex, okay? Now here's one extremely useful property for convex functions, is this; a function is convex if and only if it's convex when it's restricted to any line. That's very, very useful.

And in fact, it's one of the best ways to actually establish convexity of a function. And essentially it means the following; I already said that convexity of a function on R is a non-issue. Plot and use your eyeball. This says in principle, convexity of any function is a non-issue. Now the only hitch is you have to unfortunately plot and use your eyeball on all possible lines that pass through the set of which there are [inaudible] number.

Okay, so that's the only – but conceptually it means that there's nothing subtle about convexity. It's basically if you take some complicated horrible function, multiple dimensions, and you take a line, then – and then view that function on that line, you should view something that looks like that. And if that happens no matter what line you choose, it's convex. That's what it is.

So this is not too hard to show, and may indeed be coming up on a homework or something for you. I mean, to establish it, this is the case. And let's look at an example; so here's an interesting function; if the log of the determinant of a positive definite matrix – by the way, that's a complicated function right there.

For example, if X were 10 by 10 and I wrote this out, it wouldn't – it would take like a book to write out what that is because you'd have 10 factorial terms in log det X. So this is a log of a polynomial of these 100 by – well okay, it's 10 by 10 so there's only 55 entries because the bottom is the same as the top or whatever. So there's 55 entries, you know, free variable.

So this is the log of a polynomial in 55 variables, and that polynomial probably would take a book. I'm just making a wide guess. By the way, it could be a whole lot bigger than a book too. I could be off a bit. But – so that's – this is really a fantastically complicated function. Actually, this function is concave. That's going to be very important. It's going to play all sorts of roles later.

So let's actually show that, that this is concave using this technique. This is – well it's the only technique you know except for applying the definition. Now – by the way, what does it mean to say its concave? Basically it says that if you have two positive definite matrices, and you evaluate the log and determinance of them, and then you form a blend of the two, let's say the average.

It says, the log of the determinant of that average is bigger than or equal to the average of the log of the determinants of the end points. Does everybody follow that? So that's – and this is not an obvious fact. I mean, once you know it it's obvious, but – well it's actually not obvious, let's just say that. Even when – once you know it it's obvious only because you know it.

Okay, so let's see how this works; so to establish this we have – what we have to do is we have to pick an arbitrary line in symmetric matrix space, okay? So what does a line look like in symmetric matrix space? Well it looks like this – and it's one that passes through the positive definite cone.

So it's going to look like this; without lost generality it looks like a point. X, which is positive definite, plus T times a direction V. Now this direction V is a symmetric matrix, but it does not have to be positive semi-definite, right, because that's a line in a direction – there's no reason the direction has to be positive – it does have to be symmetric. So this thing describes a function of a single variable T.

And we have to check that this is in fact a concave function of T, so that's what we have to do. By the way, if you're looking at some function, you have absolutely no idea if it's convex or concave. First thing to do when you're near a computer, sit down, generate arbitrary line and plot and look. Look at 10 or something like that.

And by the way, if you find one that doesn't curve up then you destroy all evidence that you did this and then comfortably say, obviously that's not convex, and then erase all the files that – it's three lines, but the point is then you erase the evidence.

If they keep coming up like this, then after 50 times it's like, well okay, here it goes. And then you roll up your sleeves and you move in to prove it. And it may not help you, but it's actually quite useful to do this. I've actually failed to follow this advice on several occasions, jumped in 45 minutes later, wandered – got tired, wandered over, typed in a few, quickly turned up, found a point where it had the wrong curvature and then realized I was just – I had just been completely wasting my time, so – okay, I just mentioned that.

Don't tell people where you learned that – where you heard that. All right, so let's work out what this is; well a right X – I'm going to write this – there's lots of ways to do it, but X is positive definite so it has a square root. So I'll take half out on each side, and this will look like this, T, it's gonna look like that, times X half.

That's what – this matrix is this. But the determinant of a triple product is the product of the determinance. And you take logs and it adds and all that, so you get this because the log of X half plus log – sorry, log det X half plus log det X half is log det X. So you get this thing here.

It's still not too obvious, but what we're going to do now is this; I'm going to write the – this thing is a symmetric, but not necessarily positive – it's certainly not positive semi-definite or anything like – you don't know. Matrix, we'll take its igon expansion. We'll write this as whatever, Q lambda Q transpose, and you get this with a T there like this.

What I'll do now is I'll do the trick of writing I as QQ transpose then pull it out on either side and I'll get det Q. It doesn't – det Q is either plus or minus one, but it doesn't matter. And then I'll end up with det I plus T lambda. That's a diagonal matrix. I know what the determinant of a diagonal matrix is, it's a product of the entries, and so I get this thing, okay?

So I went over this a bit fast, but I promised I would go fast, so –yes?

Student: How did you choose the directional matrix V?

Instructor (Stephen Boyd):It's extremely important that it's completely arbitrary. So the technical answer to your question, how did I choose the direction V is, I didn't. Or arbitrarily I suppose is the actual correct adverb or whatever, okay? That's – because if V is – if I chose V in any special way then my final conclusion is not gonna hold, it's not right.

To be convex has to be every line. It has to be convex when restricted to any line through the point. Okay. Now we're in pretty good shape, because I know what log 1 plus T lambda looks like. For any real number of lambda – if lambda's positive it looks like one thing and it goes like this, or sorry, it goes like that. Nope, sorry, that was from my point of view. I did it right the first time, it goes like this. If lambda has the other sign it goes like that; either way curvature is negative, and so this is concave.

Okay? By the way, this is gonna turn out to have sorts of implications. If you're in statistics, if you've taken information theory, communications, a lot of other things like that, actually, if you've done any computational geometry, it turns out some things you know are actually related to concavity of log det X.

Things involving like entropies of ram – of galcie invariables and things like that. So actually throughout the class, when we get – when we've actually covered some material you'll see all sorts of things you know from other classes are going to start to connect to various things here.

Next topic is – this is just a bookkeeping thing. It's quite eloquent, but it – and it's actually something good to know about. It works like this; when you have a convex function, it turns out you can encode the – it's convenient to encode the domain of the function by simply assigning the value plus infinity when you're outside the domain.

So if you have a function F with some domain, then we simply – we define an extended valued extension as follows; it's gonna agree with the function if you're in the domain and we'll assign it infinity outside the domain. And, I mean, let's not worry about this here, but technically there's a different between F and F tilde.

And the difference occurs when you call F at a point outside the domain. If you call F at a point outside the domain, you get the dreaded – what's returned as the dreaded NID token, otherwise known as 'not in domain'. Of course, what would actually happen is some exception would be thrown at you or something like that, or a NAN would come back or something like that, but that's what that is.

Whereas F tilde evaluated at a point outside the domain is of F, simply returns the token plus infinity. Okay? Now what happens is then everything kinda works. So if you have a function, sort of it looks like that, and its domain is from here to here, that's a convex function. What we simply do is we simply assign it the value sort of plus infinity outside; here, that's plus infinity, okay? So it looks like that, so you make it plus infinity.

Actually everything works, including this inequality if you say – if you take this point and this point, everything works. If you draw – everything will work because this line will now have a slope going straight up and it all just works.

So this is just something to know about, it's absolutely standard in convex analysis. So it's – you should know about it. Now for concave functions it's the opposite. You encode the domain – or I should say, you encode the not domain as minus infinity in the function value, okay?

Now we get to first order of condition. First of all, just to remind you, a function is differentiable if its domain is open. I mean you could also talk about being differentiable at a point, and you'd say that it's differentiable to point if the point is in the interior of the domain, but we'll just talk about being differentiable period.

So a point is differentiable if its domain is open and the gradient exists at each X in [inaudible]. Actually, this is a bit circular and what you really want to say – you really want to say this is not – this is informal so I'll just leave it that way.

So here's the first order condition for convexity; this is very important, and actually it's a hint as to why convex opposition actually works very, very well. Here it is, it says the following; form that is the Taylor approximation. The first order Taylor approximation of F at X. Okay, that says as a function of Y. That's the Taylor approximation.

So there's F, here is this Taylor approximation, of course this is drawn in R, but in general this is the Taylor approximation. And of course what the Taylor approximation is this; it – at the point in which you expanded, it's perfect. In other words, it coincides with the function. Nearby, so near X, F – the Taylor expansion is very near, by which I mean to say, formally near squared.

So it's small – the error is small squared. So what the Newton tells you or calculus tells you is that these two functions, one is this affine function and one is this one, is that the are very near as long as Y is near X. For a convex function this thing is actually always an underestimator. So that's the important part.

It says that the Taylor expan – the first order of Taylor expansion is a global underestimator of the function. That's what it means. Now, what – let me – I mean, this is actually the key to everything. I mean it's – once you know all these it's kinda trivial, but this is the key to everything because let me explain what happens; later in the class we'll formulate all sorts of crazy problems as convex problems.

And we'll come back and we'll say, I've solved it, this is the solution. It will be some horrible complicate problem with hideous no non-linearity's, things that are non-differentiable, god knows what's in it. And someone will say, well how do you know that's the global solution? That problem is highly non-linear, it's got all – I mean, that's ridiculous.

I mean maybe it's a local solution, I'd believe that, but how could you possibly assert that in all over our 100 there's no point that does better than your point. Maybe there's some sick little region you haven't examined yet where the function miracously does better.

Everybody see what I'm saying? And it's really a preposterous statement that you're making. I mean after a while you'll get used to it, but it's preposterous. And then you said, oh no, no, because this is a convex problem, I assure you this is the absolute best there is.

And someone will say, how can you say that, it's ridiculous. This is it, and the reason is this; from local information, that's a gradient, a gradient is local information. It says, from local information you get global conclusion, which is this inequality. So just – although the inequality is not a big deal, the fact that it says something that you evaluate a gradient somewhere, that's completely local.

To evaluate a gradient all you need to do is wiggle X around near that point and see how the function varies locally. You don't have to do any big exploration far away. What is says if that function is convex then from that little local exploration you can make global bounds on that function that hold arbitrarily far away. Everybody see what I'm saying?

So not a big deal, but this is – if you ever – what's going to happen is three weeks into the class things that are just preposterous will be asserted, you'll be asserting them in fact. And if you ever wanted to go back and say, well what – you know, but you know how it always is, right, there's just a string of little trivialities and the next thing you know you've said something quite deep, and then you always want to go back and say, where exactly did that happen?

Anyway, I say it happened here. That's what I say. Okay, second order of condition; I guess you've seen this. This you've probably seen in a calculus class or something like that. Usually it has to do with sort of this part, the suficion conditions or checking if something's a minimum or maximum or something like that.

And it basically says that the – it says, if it's twice differentiable, it says it's a hessian, which is the makers of partial derivatives. If that is – it says the following; it says it's convex if and only if that matrix is positive-semidefinite, okay? So that's the condition.

And then there's a gap in characterizing strict convexity. And the gap says something like this; it says, certainly if the hessian is positive-definite everywhere then the matrix – I'm sorry, and then the function is strictly convex. Actually the converse is false and an example would be S to the fourth on R. That's a strictly convex function; X to the fourth, but its second derivative at zero is zero, okay?

That's a case where the second derivative fails to be positive at the origin. So this is the condition. So this is useful sometimes. Actually, this is less useful, in fact, and also when – although you will have to – I mean, we'll see to it, let me put it that way, that you'll have to use this method to establish convexity of something. But as you'll see later, this is to be avoided because generally it's a point to show something as positive-semidefinite, unless it's really simple, the function.

Now we can knock off some examples real quick. Let's characterize all quadratic functions right now. So a quadratic function looks like this; it's a quadratic form, it's a p-asymmetric here, a liner function and a constant, okay?

So the gradient of this function is PX plus Q, and the hessian is P. So that's the – it's constant, it's got a constant hessian. So a quadratic function is convex if and only if P is bigger than or equal to zero, if it's a positive-semidefinite matrix. And an example would be least squares objective. That just – immediately. So here the hessian is A transpose A and that's going to be positive-semidefinite so that's always convex.

Here's a function that is not obvious, it's one – this is probably one of the first functions you encounter that is not obvious, it's not obvious that it's convex. Maybe other than minus log det X, but here's one; X squared over Y just two variables. So it's – this is convex in X and Y provided Y is positive.

And let's take a look at that – first of all, let's just check a few things. If you fix Y it's convex in X that's clear. If you fix X it's convex and Y because it's a one over Y in that case, okay? Now by the way, I'm using there this idea that these are essentially lines. I mean that's a line, one is aligned with the X axis, one with the Y axis. So if a function of many variables is convex, it better be convex in each variable separately.

The converse of course if completely false; you can have a function convex in each variable separately, but not jointly convex. And the answer in – I mean, the distinction comes completely from the Hessian, right? So you have a, for example, a function of X and YQ variables, you have a two by two hessian that is required to be positive-semidefinite.

To say that it’s convex in X and Y, it says the diagonals of that hessian are non-negative. That’s what it means. To – that tells me nothing about the off-diagonals, so there’s a further condition linking the two and the cross condition. Question?

Student: Is there any way to prove convexity or show convexity from [inaudible] projecting it on to one dimension [inaudible] number of times and –

Instructor (Stephen Boyd):Be careful because we’re not projecting onto a dimension, we’re restricting to one dimension. And the answer is, yes. In principle if you restrict it to any line, any sort of one dimensional set that passes [inaudible] and the result is always convex, then it’s convex, but it has to be understood that’s in principle unless you’re prepared to do something symbolic with every line like we just did with log det.

Okay. So let’s take a look at this function, and let’s show that it’s con – by the way, here’s a plot of it, so it looks – I don’t know, some people have told me it looks like a boat or something like that, the bow of a boat or – I don’t know, anyway, so and if you look in – I guess, one slice it’s quadratic and it’s – I guess it’s – I don’t know what it is anyway so, if you slice it different directions you get obviously convex things.

So lets work out the hessian, well I worked it out, and I’m not going to calculate in front of you all the partial derivatives of the things here, but you calculate partial squared F, partial X, partial Y, and the diagonals, and you fill it out and you get a matrix which indeed is rank one, and can – and positive-semidefinite because you write it this way.

And here we’re using the fact that Y is positive here, to ensure that this is positive-semidefinite. So that’s convex, so that’s your – maybe one of your first non-obvious convex functions. Here’s another on; the so called log sum X function. That’s the log of a sum of exponentials of variables. Actually, before we go on let me just say a little bit about this function; it’s – first of all it’s sort of like a smooth max, or in fact some people call – there are fields where this is simply referred to as soft max.

So I know whole fields where this is universally called the soft max. The reason it’s soft max is this; if you take a bunch of variables, X1 through XN, then X increases of course very quickly. So the biggest X is a lot bigger. If there’s a good gap between X – the largest X and the next one, X will accentuate the spread.

Okay? If you add these up and then take the log – if for example one of the X’s is kind of isolated and far away from the second – if the biggest is away from the second biggest then you can actually quickly see that basically this sum X is basically X of the largest. You take the log of that and you get the largest. So this is sort of a smooth – it’s a soft max some people call it.

I should also mention this is – if you’re in electrical engineering, this is the DB combining formula. So this is how you combine powers that add incoherently. So if you have for example, if you have a bunch of powers adding, and the powers come in at, minus 15 DPM, minus 22, minus 37 and minus 3DBM, everyone in here knows what the power of the result is, except I forgot the numbers I just listed, but I think it’s minus three with the largest one. The answer is it’s minus three.

So, whatever that is, minus three DB reference to a millawatt. And let me explain what that is; X convert from decibels to power. This does incoherent power addition and log converts back to decibels. By the way, obviously if you’re not in electrical engineering you don’t have any idea what I’m talking about, and that’s just fine.

So I’m just saying this is a function, you’ve seen it before and it comes up in statistics as well in exponential families, this comes up all the time. Okay. So, in a little statistical mechanics too, it’s the denominate – a fancy version of this is some normalization constant or whatever. I’m sure people here know a lot more about this than I do.

So that’s a convex function, and the argument would go something like this; you take the hessian – by the way, if you’re curious, no one can look at this and write this formula out for the hessian. Just, you know, so if you’re looking at this and saying like, oh, that looks pretty easy, anyway, it’s not.

You have to sit down and first you try to find some secret rules for – chain rules for calculating the hessian of some function, that’s the first thing you do. Then that gives you a headache. Then I usually just basically – at some point you give up and you just go in there – you just calculate partial squared F, partial XI, partial XJ, there’s just no way around it. You get this huge horrible mess then you have to go from your mess, your index by index representation to a matrix representation and then you’re off and running.

Then you go back and erase all that and then write things like this. So – just like this and then that’s the idea, but lets take a look at it and see what it says; now when you do this you look at it and you see well there’s this first term, Z here is this X of X, so it’s obviously – it’s non-negative.

This think here, it’s interesting. That’s a non – that’s a positive diagonal matrix so this is positive definite – positive-semidefinite – well it’s positive-definite. And of course what you’re hoping for is to add that to something positive definite at which point you – it’s easy because you just say the sum of two positive-semidefinite matrices is positive-semidefinite, done.

And there’s a little minor problem, which is that this sign goes the wrong way, and that could mean two things; it could mean either this is not convex, or you have to work harder to show this thing is positive-seimdefinite, and it’s the latter here.

Student: Is Z a vector here?

Instructor (Stephen Boyd):Yep, Z’s a vector.

Student: What is diag of [inaudible]?

Instructor (Stephen Boyd):What is diag of?

Student:[Inaudible]?

Instructor (Stephen Boyd):Of Z here?

Student: Yeah?

Instructor (Stephen Boyd):What is diag of Z? Oh, diag of Z, you take a diag – this is actually entering – of course its mat lab slang or something. But it's entering – it's sort of entering mainstream mathematical notation.

Student:[Inaudible]

Instructor (Stephen Boyd):Exactly. Yeah. So diag of a vector imbeds the vector into the diagonal elements of a diagonal matrix. What department are you in?

Student: Double A.

Instructor (Stephen Boyd):What?

Student: Double A.

Instructor (Stephen Boyd):Double A, okay. Now wait a minute here, have you never used [inaudible]?

Student:A long time ago [inaudible].

Instructor (Stephen Boyd):It was a long time ago.

Student: Maybe.

Instructor (Stephen Boyd):Maybe, okay.

Student: Last year, I think.

Instructor (Stephen Boyd):Okay, last year. Okay, fine. Okay, that's fine, just checking. I thought you were going to say math or something like that.

Student:[Inaudible]

Instructor (Stephen Boyd):Okay, all right. So it was in a CS class last year, which – the memory of which – you didn't remember what class it was or – what was it?

Student:220.

Instructor (Stephen Boyd):220?

Student: Eight.

Instructor (Stephen Boyd):Eight, okay, I figured. Okay. So back to our thread here, it's not over yet, you still have to show this is positive-semidefinite. How do you show a matrix positive-semidefinite? You simply show that the associate quadratic form in always non-negative. So you put a V on the left to be transposed, a V on the right. You plug that in here and see what happens and you get this thing.

And this turns out to be greater than or equal to zero using the Cauchy-Schwartz inequality applied here, okay? These are not – this is not obvious. Okay? This is not something that you just type in. This is like 30 minutes of thinking and breaking pencils and things like that.

Oh, you know, I just remembered something, speaking of that. I don't know that it's been announced yet, but I – it's just a suggestion. A suggestion actually it's not a bad thing to work on homework in possibly small groups of people depending on who you are. I mean people already probably have the sense to do this anyway, but you're being officially encouraged to work in small groups like that. In which I case I could say so after 30 minutes of working this, breaking pencils, throwing pencils at the other people in the group and things like that.

Actually working in a group is very good because it's easy to think you understand something, very easy, when you're by yourself. And it turns out if you have to address like two or three other people and try to explain it – actually, you know while you're explaining it you'll realize that so your mouth is going like this, right, and there'll be something in the back of your head, I don't know what part of it is, will actually come out and say, by the way – also just looking at the others in the group you realize, this makes no sense whatsoever.

I mean, what you're saying makes no sense. So it's a very good [inaudible]. Whereas a lot of things you know by yourself can be totally obvious. Totally obvious, then you try to explain it to someone. Also, if you try to explain it and it's this long complicated – you go, look, its super simple, it's so easy. You just – okay, you do this, but then its, well hang on, I forgot to say this.

And then also, by the way, do you remember this and then you put that there – this is also a hint that this is way to complicated, your description of it, so you're encouraged to work in small groups. All right, let's move on. So a geometric [inaudible] is concave, so this is the product of a bunch of variables, these have to be positive, although actually this works for non-negative.

So it's a bunch of variables and then the enth root of that, so that's concave. And it's the same type of argument as for log sum exptH, same type of thing. Okay, some other connections, the idea of an epigraph and a sublevel set. So if you have any function then the sublevel – the alpha sublevel set is the set of points with F value less than alpha.

By the way, later in the course F will be an objective or it will be some type of objective, or something like that. And for example, if F is a design, then if F represents the power dissipated by some circuit design or something like this, this will be the set of designs that meet an alpha spec.

That's what this is going to mean, okay? That's what a sublevel set will often mean. By the way, if it's an estimation problem, an F is a measure of implausibility, like negative log likelihood in a statistical estimation problem. This will be the set of points, which are at least alpha plausible values, okay?

So something like that. All right, now if you have a convex function then the sublevel sets are convex. By the way, the converse of that is false, but it's a very important thing and we'll have a name for it very soon, a function whose sublevel sets is convex.

The real correct connection between convex function, convex set, because we've overloaded the word convex now to mean two things; it applies to sets and it applies to functions. The real connection is through something called the epigraph. So I guess epi means above, and so epigraph means everything about the graph.

The graph of a function of course, is the set of pairs, XY. So it's an RN plus one, it's the actual graph. The epigraph is everything above it. So if I have a function, the epigraph is this shaded region like this. And here's the real connection between convex set and functions.

A function is convex if and only if its epigraph is a convex set. So – and in fact, it turns out you should just be thinking about convex functions in terms of their epigraphs just always. So, for example, when I say here's a function like that, in fact let's put a domain restriction on it from there to there, you should just immediately visualize this set drawn in, and it's a convex set.

Actually this is going to be important because we're going to do a lot of calculus on – with convex functions and you want to think about what does it mean in terms of the sets? That's what you want to be doing. So this you should just be thinking of at all times, the epigraph.

Okay. Let's look at Jensen's inequality; so our basic inequality for convexity is this; it says if you take a point between zero and one, if you take two points and take some kind of weighted average of the two, that's what this is, and evaluate it, it's – so this says that F of the weighted average is less than the weighted average of F. I said it right. I always forget this.

I'll show you a pneumonic in a minute, which you can sneak off and do if you have something to draw on. You can even do it if you can't draw, but it takes me longer. Now there's tons of extensions of this, for example, instead of just two points you could have a finet number of points and a bunch on fada I's that add up to one and are non-negative. That's a convex combination.

And the same inequality would be true thinking of an accountable infinite number of points, some combination that way, and then you can have an infinite one. The most general is this; if F is convex, then F of an expected value is less than or equal to expected value of F of X. Now that's called – that's Jensen's inequality.

And this is where Z is a random variable, however, which is in the domain of F almost surely. So that's this, and this is Jensen's inequality. And this basic case is nothing by this; it's a distribution on Z, extremely simple. It takes only two values, X and Y with probability theta and one minus theta, and you recover this thing.

So this is Jensen's inequality. I never remember Jensen's inequality, especially because whenever it comes up usually F is half the time convex, half concave. Of course if it's concave it goes the other way. And so I actually never remember it and I usually have to just draw a picture and remember this thing. There's another way to say it though, I know what that is; so if you want a general pneumonic about how it works, it basically says that for a convex function dithering actually hurts.

And let me explain what that means; it means that if I have a function – a point here, and here's a convex function, like that, okay? Let's imagine, that's actually my target point in some process, but now, actually when I manufacture it, I actually get a distribution of values like that, okay? Whose mean is this point, everybody cool on this?

So that's the – this happens, so this is – and then this tells you the cost, okay? And this could be the power, I don't really care, something like that. Or the speed of some – lets make it the power. So this is the power, and it basically says, you know, look, these points, that are where the manufacturing [inaudible] they were on your side. The threshold voltage went up or whatever, something happened, and it actually dissipates less power than your nominal design.

Everybody see that? And this is where it worked out badly. And now I ask, what's the expected value of the power? What is it? Well the first thing you would do is you'd say, well look, it's around here, because if I approximated this by an affine function and I propagated this distribution through an affine function, then its mean would be the same. So the first quick answer is something like this; well it's the same, if this is like one watt, and you have these manufacturing variations, sometimes you're less than a watt, sometimes you're more.

And it says that the average is going to be on the order of watt, you know, like a watt. Everybody got that? But the fact is its worse – the average is worse than a watt. It's like 1.05, it can't be better. And the reason is this; it's true that when you go up and down here, The first order approximation, in that case sometimes you're less than a watt, sometimes more, but because the curvature is upward you get – let's see, you pay more when it's bad then you recover when it's good. Did that make any sense? And that's entirely due to the curvature. So that my pneumonic for Jensen's inequality. It basically says manufacturing variation generally isn't good. Yes?

Student: What [inaudible]?

Instructor (Stephen Boyd):That's your question? So this is a good time for me to announce this. There will be periodically times, I'd say multiple times per lecture where I'll go in something and I'll make no sense whatsoever. Its best – if this starts happening like three or four times a lecture then you can let me know, but that's good feedback.

It's best at least for now in the spirit of just rushing forward, we'll just move on. Did anyone understand what I said? A handful of people, they're just probably just being polite. Okay. You'll learn to just move on. Now, let me – actually, let me – this is a good point to kind of explain – give a sign post and say where we are and how this works.

What you're gonna have to do is you're gonna have to look at functions and figure out if they're convex or not, that's what you have to do. So the methods I just looked at involving lines and all that, if you have to resort to that, if you actually write out a hessian, I mean this is to be avoided to be honest with you.

This is – you do this only in a last resort. Of course every now and then you have to; we'll arrange that you will have to because everyone should have to do this once or twice or something. But the point is the right way to do it is using a convex calculus. And so the right way to do it is this; what's gonna – this method three, just sort of just general method is gonna work like this, you'll learn a bunch of atoms.

You've already seen a bunch, affine function, powers, log sum exp, X squared over Y, norms, quadratic functions, thing like that, where once you know it, minus log det X you know it's convex, okay? Then we're going to look at a calculus. And a calculus meaning methods to combine these and rules for showing it's convex. And you saw this with convex sets last lecture.

But here there's gonna be a bunch – there are gonna be some simple ones here. Now in this rule set, these divide into what I would call sort of the really obvious basic ones and then there's sort of that intermediate tier, and then you get into the advanced ones and the ultra advanced ones and things like that, there's sort of no limit on these things.

So I will tell you when we move into different levels of esoteric on this, but – okay. So these are extremely basic sense of following. If you have a convex function and you scale it by a non-negative scalar, it's convex, that's totally obvious. If you add two functions that are convex, it's convex, okay? And that extends to adding five functions and it goes to an integral or even an expected value of convex functions would be convex.

Composition with an affine function; so if you pre-compose with an affine function, so in other words if you apply an affine function then a convex function, you get something that's convex, and it's – many ways to check this. Actually, just directly is simple enough, just with the old theta in there. Okay? Very simple to show these things.

Let's look at some examples; we can make some examples now. F of X is minus log sum VI minus AI transpose X, and this of course is defined on the region where these are strictly positive. That's the interior of a polyhedron. So I have a polyhedron defined by AI transpose X less than VI. The interiors where that's strictly less, and that's where this makes sense.

This is by the way called the log barrier for this polyhedron, and you have a minus sign here. So you know, that's kind of a very complicated highly non-linear function of X. You could work out the gradient; you could work out the hessian. This one wouldn't be too bad because if you work out the hessian, which I would not recommend doing by just getting in there and slogging it out, calculating partial squared F, partial squared XI XJ, but using some of the rules for calculating hessians, some of these are in the – these are in the appendix by the way.

It would work, but the easiest thing by far is to say this, here's my function, ready, I'm going to apply – here, I'm going to take this function which is fi of Z is sum minus log ZI, okay? So that's it, it takes a bunch of variables, takes their logs, takes minus sum. That's a convex function. Why? Well each of these is a convex function and a sum is convex. That's a convex function. Again, nothing stunning yet.

Now we're going to simply pre-compose this function with this affine mapping. I'll call it B minus AX, that's an affine mapping. And that gives you this function and then here, I just applied this, this composition rule. Here's another one; is the norm of any affine function, so norm AX plus V is gonna be affine – convex, sorry. It's going to be a convex function of X.

Okay. Here's one that maybe not totally obvious; so here's one convex function and here's another one, like that. The point wise maximum is this function here; it looks like that, okay? So at each point it's the maximum of the two – of the functions, okay? It looks like that.

By the way, in terms of epigraphs, what does this correspond to? Precisely. So calculating point wise maximum of functions, in fact you can even write some silly formula for it, you know something like this; epi of max over I FI is equal to the intersection over I of epi FI, something like that, okay?

So that's the correspondence here. That preserves convexity, okay? And you know, it means for example, here this function which is the max of a bunch of affine functions as a Piecewise linear function, that's convex. Obviously not all Piecewise linear functions are, but any Piecewise linear function expressed in this way is convex.

Here's one, this is again, not obvious. The sum of the R largest components of a vector. So take a vector in R50 and the sum of the top three elements. That's a very complicated Piecewise linear function, it's convex.

Why is it convex, because it's the maximum of A transpose X or A – this is for the sum of the top three, is any vector with three one's and 47 zeros. Now there's a giant pile of those, there's 50, choose three. But the point is, it's the maximum of 50 choose three linear functions done. It's convex.

So you have to watch out here because you slowly sneak up on this and you'll find out after a while you've actually done something and some of these things are not obvious. Proving this by another method would really – could be very, very painful. I mean, this one, hessian doesn't even exist, it's not even differentiable, which would save you the horrible trouble of getting in there by hand and working out a hessian.

But just proving that directly would be a real pain. Now this maximum business extends, and it extends to an infinite max or a point wise supremum. So here's the statement; if you have a function of two variables, X and Y, and suppose it's convex in X for each Y in some set, you don't even know what the set is, totally irrelevant what the set is.

Finide, infinite, makes no difference whatsoever. Then it says that if you take – again, you can leave this as max if you haven't seen this before, this is simply – if you simply take the maximum over this possibly infinite collections of functions point wise you get a new function, that's going to be convex.

And here's some quick example; lets look at a couple of – let's start with this one, let's take the maximum eigen value of asymmetric matrix. Now we discussed that before, that is a really complicated function. For a matrix that is six by six or bigger, there is no formula for this, none, because there's no formula for the roots of a sixth degree and order and higher order polynomial.

I mean, you don't need to know that, but it's a good thing to know. Well it's not useful; it's just a cool thing to know. This is really a very complicated function, the maximum eigen value of a matrix, but watch this. If the supremum of Y transpose XY over all Y that have – over the units sphere. The units sphere is the set of all points whose norm is one.

Now let's check me and see how the argument goes. Look at Y transpose XY. Now by the way, when you look at that you are – at this point you are trained and wired, all by the quadratic part of your brain should be lighting up. This has been proved in FMRI studies, okay? But the point is actually here the variable is capital X. What kind of function of capital X is this? Remember, you have to suppress the flashing quadratic neurons. It's linear, that's a linear function of X. This is sum, XIJ YI YJ, it's linear in X.

Okay? So for each Y that's a linear function. This is a supremum of an infinite collection of linear functions. In fact, there's one for every point on the units sphere in our end.

Supremum of a bunch of linear functions, linear functions are convex. Supremum over these things is convex, that's a convex function. And now you know something that's not totally obvious.

I mean, it's not that unobvious either, but it's – this would be – since that's like a two line proof or something like that, that's not bad. And of course it's a disaster if you actually try to write out what lambda max is. I mean, just – you write down square root of – or that's not lambda max, but if you write down the largest of the lambda I's for which the character [inaudible] vanishes, it's all over, you'll never recover, this is an example.

Let me hit the next one, is composition. So under certain circumstances composition preserved convexity. So let's see what that is; if you have H of G of X, -- so the rule goes like this, and I'll – the famous one is this, it says that a convex increasing function of a convex function is convex. Okay?

So if – this thing will be convex if you have a convex – if the outer function is convex and increasing, which – I mean non-decreasing, okay? So that's the condition. And the way to derive these as other ones, for example, it's convex if this thing is concave and H is convex and non – and decreasing, roughly is what it is.

Now the way to check these is simply to write out the chain rules. So if you take – you imagine that these functions are differentiable and of one variable and you work out the second derivative and you get something like this. And from things like this, this is how you read off the rules, by the way, these things can – unless you're doing this all day long these rules, you can easily forget them.

What you should remember is there are composition rules, and you have to go back and [inaudible]. Whenever I'm somewhere and I have to figure this out I actually quietly write this out and then figure out the rules myself. By the way, the rules hold even when these are non-differentiable. You don't need any derivatives. This is just to check.

And lets actually – maybe we can make up a new rule, let's make one up for fun, ready? All right, let's see, let's try here, lets say that G – I'm not gonna remember what I'm saying, so I'm gonna write it down. Suppose I told you – I have to get this right – suppose I told you that G prime is positive. So G is going to be increasing, and I told you – do I even need G prime? No, sorry. I take that back.

I'm going to tell you that G is concave. So that means this here, and lets make – lets see if I can get it right, and lets – let me ask, what are the conditions on H to make F concave? Does that make sense? So what we want is – we're gonna say that that's – we're gonna assume that that's less or equal to zero, and I need thing to be less than or equal to zero. Well that means this has to be positive. So I have to have H prime positive. So H has to be increasing.

And then I look over here, that's positive no matter what, so I need this to be less than or equal to zero, and I just made up my very own new composition rule, and it's this – I hope I get this right. Okay, let me go very slowly, if H is concave and increasing and G is concave – okay, ready, so a concave increasing function of a concave function is concave. Isn't that right?

Anyway if I got it wrong, I don't care, you get the principle. Okay, it's a 300 level class, I can mess up minus signs all I like, that's your job to fix them and stuff like that. So I think I said that right. So let's look at some example – by the way, the only one I actually remember, to be honest with you is a convex increasing function of a convex function is convex.

Then I have – I remember one other thing, there are composition rules and then I have a note attached to that, there's lots of them and they get very confusing, although there are people, I've noticed who just know them immediately. They'll say, oh yeah, not that's a concave increasing function – no, decreasing function of a convex function, that's con – something or another and I'm like, really? I don't know. Then they have some internal pneumonic or something for doing this, but I don't know them just to tell you the truth.

Let's look at some examples; the exptH of a convex function is convex. And that's by the way is very interesting. It says that the exponential actually can only – it preserves positive of a curvature. If a function curves up, [inaudible] also curves up. That's what it's saying geometrically.

Inverse is interesting. It says the inverse of – it's not the inverse, it's the reciprocal, there you go, that's the English word, the reciprocal of a positive concave function is convex. So, for example, let's see if I can get an example of that. One over the square root is a – that would be a convex function, and indeed it is, it's X minus one half, and it goes kinda like that – sorry, for you, like that.

Actually, by the way, that's fine because flipping the axis doesn't change the curvature property, so I don't have to draw it for you. If it looks convex for me it looks convex for you. So that's the composition one. There are then vector compositions ones and I'll say – I'll give some examples here. So here you have vector composition, so here I have a function of – a multi-argument function of a bunch of other functions.

And now, of course, the possibilities just explode because you have horrible things where you have a function that's sort of increasing in some components, decreasing in others, the arguments themselves are either convex, concave, and it gets very complicated.

But here you have something like this; X is convex if all of the functions are convex, that's the easy case, and H is convex and decreasing in each argument – sorry, non-decreasing, roughly speaking increasing. By the way, there's one subtlety here that I want to point out, there a tilde here.

Tilde is the extended value extension. And I'm not going to spend time in lecture going over this, but you want to read that part of the book at some point about this tilde because that's not a typo, it's very important.

Let me give a couple of examples here. By the way, what this shows is that actually the earlier rules that you learned are actually – they can be derived from this. Let me give an example; how about the sum of convex functions? So here's a function H, H of Z is one transpose Z, it's the sum of the Z's. Okay?

This function is – well let's work it out. It is certainly convex, right, in Z? It's also increasing in each argument, do you agree with that, because it just sums the I. So it's obviously increasing in each argument. Therefore, by this composition rule, it says that if I compose this with a bunch of functions, each of which is convex, the result is convex.

So I – so from this rule I've rederived the simpler rule which is the sum of convex functions is convex, everybody see that? Let's try one more. Lets try H of Z is the max of Z. Well the max function is itself convex. It's also – it's increasing in each argument, I mean that's clear, right?

If you increase any element of a vector the max doesn't go down, so it's non-decreasing. Therefore, this subsumes actually several of the earlier ones. Actually it turns out there's only two rules. There's this rule – in fact there are only two rules, there's this rule and there's the affine pre-composition.

But it's good for a human being at least to think of them as eight or 10 rules or something. Quick question?

Student: Increasing [inaudible] do you mean also that when you – that they increase from vector to vector, or just that – [Crosstalk]

Instructor (Stephen Boyd):Nope, I do not mean that. I mean this – I mean, hold all element fixed except one. Increase one the function cannot go down. That's what non-decreasing in each argument means. Okay, so I think we'll quick – this covers, these are the basic rules. By the way, this was very fast, [inaudible] we'll cement this, don't worry, and we'll continue next time. And then maybe even by next week it'll get interesting.

[End of Audio]

Duration: 77 minutes

ConvexOptimizationI-Lecture04

Instructor (Stephen Boyd):Great, I think we're on. Let me start today with a couple of announcements. Let's see, the first one – you can go down to the pad, is that we have posted homework 2. So – which I'm sure you'll be very happy to hear about.

And homework 2 by the way is – I promise, sort of the last one like homework 1 that's this kind of slog through interminable math. So it'll be the last one. And then after – and starting homework 3 I actually promise it'll start getting interesting. So that's our promise.

Let's see, the other announcement – I guess this one's important; next Monday there is a holiday. When – that's normally when the section is. There's gonna be a section and we are trying to figure out when it's gonna be.

We've been offered two, like not optimal times. I think one is like Tuesday – next Tuesday at 11:00 a.m.

Student:[Inaudible]

Instructor (Stephen Boyd):Wait till you hear the other one. Wednesday – oh, I don't remember this one – I don't know, I can't remember anymore. Kwangmoo, do you remember? Wednesday at 2:00 p.m.?

Student:2:15.

Instructor (Stephen Boyd):2:15.

Student: Tuesday.

Instructor (Stephen Boyd):Now Tuesday, what do you mean?

Student:[Inaudible]

Instructor (Stephen Boyd):Wait, well we have random – okay, does anyone have like, I don't know, violent or other – can articulate a very good reason why it shouldn't be either Tuesday or Wednesday? Our thought was that Wednesday was pretty close to the homework due time or something like that.

So Tuesday – Tuesday at a crappy time. It's also in a crappy room by the way also, so Tuesday crappy time, crappy room. Sure, okay. Hopefully that hasn't been taken now, so we'll – this obviously we'll announce – we'll announce this on the email list, and we'll also of course put it on the website, so when we – when that's fixed, but there will be a section that would have been Monday, it'll be Tuesday.

No, wait a minute; I can announce it in class, so – which is strange. We're gonna repeat that when there's another Monday holiday later in the quarter, and we don't feel like missing sections, so whatever happens next week is going to happen one other time during the quarter.

Okay, another – another announcements, my office hours, I had promised to expand them, modulo quals, and in fact I – it looks like, you have to check though, it looks like I actually will have some time today after class, and then also after 3:30 today, so I'll be around if people want to come by.

And then sometime of the weekend, or whatever, I'll figure out when my actual office hours will be. But I'll put something – there'll be something in the afternoon, like maybe Tuesday afternoon or something like that, and I'll coordinate that with the TA office hours.

And then that'll of course be announced. Okay, any questions about last time, our whirlwind tour through – or actually really like a sprint through convex functions? If not, we'll finish that up today. Is it – a ton of material here and the only – as I said last time, the important part is that you should get a – just sort of the basics, get the basics down and then some of the more advanced stuff you'll get when you're forced to use it.

Student:I have a question.

Instructor (Stephen Boyd):Yes?

Student: When we were talking about matrix space and restricting it to a line to determine the [inaudible] –

Instructor (Stephen Boyd):Right.

Student: How do you ensure that when you add a multiple, to make sure you can stay in that same space?

Instructor (Stephen Boyd):Oh, well the – right, so the question had to do – we were analyzing convexity of a function of matrices, which I forget what it was. I could look back here, but I won't. Of symmetric matrices, and we had to generate an arbitrary line in symmetric matrix space. And we did that by picking a symmetric base point, that was some X0 or – I don't know how I labeled it, and then a symmetric direction, and that was this; X0 + t V.

Now it's a sub-space, or quite clearly if you add a symmetric matrix to a scalar multiple [inaudible] you get a symmetric matrix. So this is a symmetric matrix. Now there we also assumed, without loss generality, that X0 the base point, was positive-definite because I think it was the log get maybe we were looking at, or something like that.

This was assumed to be positive-definite, but V was not, V was a direction. I don't know if I answered your question. I think I just said what I said last time. Are you – how's your cognitive state? Is it improved?

Student: So what you're saying is when you add matrices of the same type as that one, you stay in this?

Instructor (Stephen Boyd):It's a sub-space, absolutely. A set of symmetric matrices is a sub-space, ad a linear combination of – a linear – a multiple of one to another, you're in the sub-space. Okay, let's move on. And let me just remind you where we are – what the big picture is.

The big picture comes down to this; you're gonna want to – well two things, you're gonna want to construct convex functions, but you're also gonna want to deconstruct convex functions. And what I mean by that is that there'll be some function that's gonna come up in some problem, I don't know what it is, you know, medical imaging or optical control.

It doesn't really matter. And you're gonna want to know is it convex or not. And in very rare cases, you will be forced to do this by hand. So by hand means you whip out your theta between 0 and 1, two arbitrary points, and try to argue this inequality.

That's gonna occur occasionally. If the function is twice differentiable, the equivalent of doing it sort of by hand – and by the way, every now and then you do have to do things by hand. So to – you will actually have to work out the hessian, and it might be easy, it might be not easy to establish that the hessian is positive-semi-definite.

But in fact, the way you really want to do things is reserve these two for when there is no other option. What you really want to use is the calculus of convex functions to establish convexity. And for that matter, to decompose a function in front of you into a sequence of operations with the leaves of the tree sort of being adams, and adam is a function known to be convex or concave or something like that.

And here you want to know about the operations that preserve convexity, and we look at sort of the really obvious simple ones, you know, nonnegative weighted sum, that's kind of obvious. Composition if an affine function, that's also very easy to show. Pointwise maximum supremum, we looked at examples too, and by the time you get to sort of supremum, you can actually start showing convexity of some things that at first glance look pretty complicated.

I mean, even with maximum, for example, to show that the sum of the five largest absolute values of entries of a vector is convex. It's not – first of all, it's not an obvious fact, it's not that easy to show. It's like one line if you use a pointwise maximum argument.

Then we look at composition. That's where we are now and I'll finish that off now. And I pointed out that in fact, these are – this is not a minimal list. And in fact, the minimal list, as far as I know, of the things covered here, this is basically like two. There are two operations that are used because composition actually subsumes nonnegative weighted sum and pointwise maximum supremum.

All right, so let's look at composition. The composition, the most basic case, it's the only one I actually remember, is the scalar case, so here you compose a function with another, and the question is, when does that preserve convexity?

And the answer is, and this is the only one I remember, is a convex increasing function of a convex function is convex. And here I should – I do have warn you, you should – I mean you're gonna have to know that there is a subtlety. This is not a subtlety interesting for its mathematical interest or anything like that. This is a subtlety that comes up in practice, and will mess you up if you're not aware of it.

And the subtlety is this; the – when you say an increasing convex function, the increasing actually refers to the extended value function. Okay? For example, you have to distinguish between for example, X squared – now X squared is clearly not – X squared on ours, obviously not increasing.

However, X squared on R+, if you restricted to R+ you would say, well that's increasing. And in fact, any normal person would say it. And it is indeed increasing, however; remember that if you simply – if you define it to be undefined for –X then the extended valued version actually looks like this.

And it actually assigns a value minus infinity. At this point, this function is by no means increasing, I mean, not remotely. It starts out at + infinity, and then dives down to zero. So that's non-increasing. Now there is another version of this, and that's this; you take the square, like this, and then you assign it to be zero below for X-, okay?

This thing in increase – this thing is actually increasing on – the extended valued version is increasing, this function. So these are things unfortunately you do have to know, that do come up, you can get deeply confused, and it can actually have real consequences if you don't – if you're not sensitive to this.

There's some irritating things here, let me just show you one. Here's one; let's suppose we want to look at something like the norm of AX – B to the 1.62. Don't ask me why, but we do. And we want to argue that that's a convex function of X, okay?

Well you can work from the inside out and you can say, look, that's an affine function. That's a convex function of an affine function, and therefore norm AX – B is convex. Now what we want to do is say something like well this is a convex function to the 1.62 power. And I'd like it to say a convex function of the 1.62 power is convex. That's kind – it's sort of true, but sort of like not true, or something like – you have to be very careful now.

All right, now I can't just define – let's get my notation consistent, H. I can't just say H (Z) = Z to the 1.62 for Z+. I mean, I can say that, but the problem is this function here – I mean, this is by the way, slang or – not quite slang, it's an informal way to say that the domain is non-negative Z. That's what this is.

You cannot apply the composition theorem, absolutely cannot. So the correct way to do it is to say this, there you go, is to define the function that looks like that. So that's the – and I know this is a little bit silly, but this is very important, and it will mess you up if you don't – if you're not sensitized to these things.

This is the function which is 0, if you pass a negative argument to it, and it's the number to the – it's the argument to the 1.62 power if you pass a positive argument to it. That function is actually increasing and convex on the extent – its extension. Its extension is. And now I can apply this to this and then I can make another argue – I can say, oh hey, look at that.

What I'm passing to the function H is actually always nonnegative, it's a norm. But if you + -- if you add – therefore, this part, this case, or this switch in the definition of H never occurs. And so indeed, we find that this here is convex. So I don't know if that makes sense, but that's it.

So I didn't mention that last time, but I thought I'd just quickly mention it now. So let's look at composition; the sophisticated version is the vector composition virgin – version here. So if you have vector composition, it means the argument G is actually multiple functions.

So let's say it's G1 through Gk, so I wanna – I'm gonna take a vector function, it takes many arguments of a bunch of functions, and again, the base composition theorem, or the canonical one that I remember is a convex increasing function of a convex function is convex.

And by increasing I just mean non-decreasing, so it's just informal. So a convex increasing function of a con – of convex functions, I should say, is convex, so that looks like this. And the way – now when you prove these, making no – I mean its three lines, right. You make no assumptions whatsoever about differentiability. But the way to sorta understand what happens is to write out the formulas.

The same for the composition case, the scalar composition. So here we will take N=1, so the second derivative of F is = to – this is just a chain rule for the second derivative of this thing, it's the – its G prime, that's of course – that's DGDX which is a vector times then the hessian evaluated G of X times G prime, so that's a quadratic form, and then plus this thing.

And that's nothing but sum. This is partial H, partial whatever, it's I component, evaluated at G of X and then times G prime prime I of X like that, okay? And now you can see for exam – we can check things out.

If G is convex this first term is nonnegative no matter what, and then the second term to be convex you need – sorry, that if H is convex, you need the G – if all the G's are convex, it means this vector is an element wise nonnegative vector that interproducted with – I don't know if that's a verb, but it is now. You interproduct that with the gradient of H.

If H is increasing that's a nonnegative vector, and so this thing is positive too. But now you can see that there's actually other ways to constructing. So for example, a more sophisticated version would go something like this; let's see if I can get it right. A convex function of a bunch of others is convex provided – okay, now let me see if I get right, I'll start at this thing, provided for each argument either the argument itself is convex and H is increasing in that argument, or the function – the argument is concave and H is decreasing.

Did that – I think that came out right. By the way, there are people who know these things and memorize them – not memorize them, but they have very good pneumonic or something like that. I'm not one of them, I just redo this. Whenever I see it – whenever I have to deal with it I go somewhere quite, or find a little piece of a whiteboard or paper and just write out formulas like this or something.

So this is vector composition. And we looked at some examples the last time. One more, and this would pretty much finish off the basic operations that preserve convexity of functions, and this is minimization.

Now by the way, we've already seen that max – there's a sense in which maximization – there's not a sense, sorry, maximization preserved convexity. And we've already seen that. That's the maximum pointwise supremum. Notice that in the case of maximization, there is no requirement of any kind on how the variable you're maximizing over enters into the function. Everyone know what I'm talking about here?

The point is that if I have a function of two variables, F of XY and I want to look at something like this, let's say, and define that – I'll try to get my notation right, as G of X, this thing, the only requirement here for convexity of G is the following; is that for each Y this function is a convex function of X, period.

That's the on – and for that matter, this thing does not have to be convex, it doesn't even have to – this doesn't even have to be like an RN or even be a vector space, it could be an arbitrary set. So supremum over a completely arbitrary family – so if you maximize, well assuming you can – when you maximize there's no condition – I mean, nothing matters as to how Y enters into this function.

Now minimize also works, and that's a bit a weird because generally what you're gonna find out with all these things are they weird asymmetries. In other words, it's – that you can do one thing but not the other. The cool part – well why we have this class is it turns out often the thing you can do is the thing you often want to do.

That's – otherwise this would be interesting, but not useful. What's interesting here is that there – you actually – it's very rare that you can sort of both maximize and minimize something like that because they're sort of like – the have opposite flavors. But in fact you can, but it's a much stronger condition, here's the condition; you can minimize – some people call this partial minimization, but the way, because full minimization would be to minimize F of XY over X and Y.

You can minimize over a variable provided – and the result will be convex provided – this is very important – F is jointly convex in X and Y and the set C over which you minimize is convex. Okay? And then this is easy to show and all this sorta stuff.

Now let's look at some examples, here's a very basic one and it's one you may have seen before anyway. Let's take a general quadratic form and block it into two – block the variable into two groups, X and Y. So this is a – that's a general quadratic form. It's the quadratic form associated with this matrix A B B transpose C block.

All right, well we know that F is jointly convex in X and Y provided this full block two by two matrix is positive-semi-definite.

By the way, when is this function in X for any Y, what's the condition, on A B C D?

Student:A is positive [inaudible].

Instructor (Stephen Boyd):Precisely. It's only that A is positive-semi-definite. There's no condition whatever on B and C. So if I were to take this function here and maximize F of XY over Y, the result would be convex provided only that A is positive-semi-definite. Yes?

Student:[Inaudible] on what condition on A could you then state that that function was not convex?

Instructor (Stephen Boyd):On what condition on A – oh, which function? The –

Student: That one.

Instructor (Stephen Boyd):F, okay, in this case because it's a quadratic form, it's necessary and sufficient. So if this matrix here has one negative Eigen value or more, it's not convex period. And not only that, we can easily produce points – I can make you a line where instead of going like that it will go like that. Actually, you tell me the line? You tell me the line?

Student: It needs to go like that?

Instructor (Stephen Boyd):Yes. You have a quadratic form with one negative Eigen value.

Student:[Inaudible]

Instructor (Stephen Boyd):Yeah. Take the Eigen vector, you sail that along that line and it'll go like that, it'll go down. And that pretty much ends the discussion on convexity for that function. Now if I minimize this function over Y here – and what we'll do is – this is a technical condition. This actually works with C positive-semi-definite.

In fact, I'll tell you what happens when C is only positive-semi-definite. When C is positive-definite, you minimize over Y, yeah, you can do this analytically, because then you take the gradient and you set it equal to 0 and all that.

And you end up with this; you know, you find the optimal Y, you back substitute that into this form, and what you're gonna get is this thing. X transposed [inaudible] A – B C inverse B transpose. Now this is the sure compliment, so – which you maybe have seen before, if not, I assure you on homework 3, is that right, yes, homework 3 you'll see it, because it's just a good thing to know about.

So that's called a sure compliment of – maybe of C in the big block matrix, so that's the sure compliment. And that's convex, and you know what that tells you, that tells you this matrix has to be positive-semi-definite. So you may have seen this somewhere in some other context, maybe estimation or – I don't where, but it would come up. It would come up in mechanics and come up I all sorts of things, this sure compliment here. Yes?

Student:[Inaudible] the graph is convex?

Instructor (Stephen Boyd):Joint convexity means that if you plot sort of X and Y, the whole – then its bowl shaped the graph.

Student: Okay.

Instructor (Stephen Boyd):Convexity in X means that if you fix Y and sort of look at a slice, it would go like this, it would go up, and it could be quite different, right, the two. Here's one; the distance to a convex set is gonna be – the distance to a convex set is convex. By the way, that's a rather complicated function. I mean, it's not simple.

If I draw some convex set like this, lets just get a rough idea of – I mean, the set is everything in here, let's work out what the distance to it is? What's the distance in here? 0, now I can draw the level curves of the distance, so for example, I might dra – I might end up – I won't do this right, but you know, you get the idea. It would look like – something like that, right?

These would be all of the points with a certain distance from here, and then as you move out, actually of course these get smoothed out, right, so if you have sharp corners in a set then the distance level curves get smoother. Like that, so you'd get things that would look like this, let's see if I can do it right, that's like actually circular. Is that it? Yeah. So I think that's it.

This is to give you a rough idea. So this function, though complicated, is convex. And the argument would go something like this; you'd say, well let's just take a look at this thing. I look at norm X – Y. Here I take norm X-Y, I have to make a – this is a very important argument, that's jointly convex in both X and Y.

That's the critical part that allows me to do this, okay. So that's gonna be convex.

Student:[Inaudible]

Instructor (Stephen Boyd):Oh, here, yeah, let me give you the full proof, okay? I mean, I could rip out – I could get out theta's and stuff and do it by hand, but it don't want to, show I'll show you the snappy proof. Snappy proof goes like this; I have two variables, X and Y. What kind of function is X-Y of X, Y?

Student:[Inaudible]

Instructor (Stephen Boyd):Yeah, it's affine, sure, but it's linear, which is stronger than affine, but affine is good enough, you're absolutely right. Therefore this is a convex function, the norm of an affine function of X and Y. That's better than getting out – whipping out the theta, wouldn't you say?

Student:[Inaudible]

Instructor (Stephen Boyd):What's that?

Student:[Inaudible]

Instructor (Stephen Boyd):It's actually liner, but affine is good enough to establish convexity. We'll look at the last example of – this one is sort of on the borderline between sort of basic methods and more advanced ones, although it actually comes up in more applications than you think, that's the problem with these things, is you look at something, think it's esoteric, and then by the time you look at enough problems you realize it actually does come up.

So that's the so-called perspective of a function. So the perspective of a function is a transformation. It does this; it takes a function from RN to R and it's gonna produce a function which has one more argument, so it's gonna be a function from RN plus 1, so if you want to explicitly make it a pair, RN cross R to R, and it's this, it's G of X and T is T F of X over T.

So it does a weird scaling, it scales X first and then it multiplies by T, okay, so that's the – and that's a point – that's a function jointly of X and T. So obviously for any fixed positive T, this is obviously a convex function, because, you know, X over T is linear, therefore, affine. F is convex and then you multiply by a positive constant, that's clearly the case.

Maybe it's less obvious that when you fix X, TF of some vector divided by a constant is convex, maybe that's less obvious, but – and showing this is not hard – by the way, the – as you might guess, the perspective of a function is related to the perspective of a set, so – and there is some connection between the epigraph of – this is sort of – you remember that one of the – probably the only maybe not obvious transformation of sets that preserves convexity was the perspective transformation.

So not surprisingly, the perspective of function is related through epigraphs to that, but I won't go into the details, we'll just look at some examples. Here, X transpose X, that's the sum of XI squared. Well that's obviously convex, but that says that T times X over T transpose X over T, which when the T's cancel out is X transpose X over T is convex, okay?

So there you go, your quadratic over linear function is convex and we got it by a quick perspective operation on just the norm. So, we went from – we took something that was completely obviously convex, and then it was something where this is less obvious.

Okay, so you start with negative log rhythm, and then you form T X – T F of X over T and you end up with T log T minus T log X and you actually get that this in convex on X and T. Some of things you know, like if you fix X for example, this is linear in T so that hardly matters and you get the negative entropy over here.

It's not so obvious that this is jointly convex in both X and T, and by the way from this you can derive the various things like negative entropy, cobac libeler divergent and all that kind of stuff. Another example would be something like this; if F is convex then G which is an affine – this has to be positive – affine positive function multiplied by F of a general linear fractional transformation, but with the denominator required to be positive, that's going to be convex, so – and that's on the – that's where this is in the domain of that – or well, sorry, when C transpose X plus D is positive and this image point is in the domain of that, so that's gonna be convex.

By the way, I recognize fully that none of this is motivated and all that kind of stuff, so don't think I don't. We'll look at a couple more concepts; one is the conjugant function which is going to get heavy use later – later in the class, so – but actually not for a couple of weeks, but then it will get very heavy use.

So this is just gonna be your first glimpse at it [inaudible]. Oh, I should add that all the things we're doing are complete – these are completely standard things, these are not weird esoteric things, they're – like the conjugant, everyone knows what it is. It's got slightly different names, some people call it the finshell conjugant and there's also some other names for it, depending on how fancy you want to sound when you use it, but these are just completely standard concepts here.

I'll let you know when we deviate into something that's not – so take any function, convex or not and you define the conjugant function, that's F star of Y, and that's the supremum of Y transpose X minus F of X over X. And so let's see if we can figure out what that looks like graphically.

It looks like this; you form the graph of F here, and then you form this function, YX – well we'll just do this in one variable. So, I form a function where the slope is Y, so that's Y, the slope here. And then it says I wanna maximize the deviation between these two functions.

Well, one side of deviation – anyway, I want to see how much smaller does F get than this line? And here you would see that that point occurs like right here. And that's gonna occur, by the way, when this is – if it's smooth, when this thing is tangent. Either way, it's gonna curve when it's tangent to this graph here, like that. And then this point will give you the conjugant function here.

So this gives you a very good way of working out the conjugant function. For example, if I decrease Y slightly, what will F start of Y do, or to be more convenient, let me ask you, what will minus F star of Y do for this function? This just requires you to visualize. If I decrease Y slightly, what happens here? Well I changed the slope, okay?

And then I go down here and I'll make a point of tangency, it'll shift down a little bit, right, or to the left – well down and to the left. And that – this point of intersection will creep up a little bit, everybody agree?

So in fact, minus F star will increase slightly, so F star would decrease. Now by the way, what happens as this slope changes? There's a critical moment here, and the critical moment occurs right there, everybody see that?

So at this very critical moment something interesting happens. I draw a line here and that would sort of give me that value of the conjugant there, negative value. Now as Y decreases further, what happens? This gets interesting. What happens is the point of contact leaves here, shifts over here like this, and this thing now starts going down. Everybody see this?

This is your first chance when we do duality, you'll get tons and tons of chances to do visual exercises like this, but this is your first exposure to this idea. So this is the conjugant – now here's an interesting about a conjugant function; it is convex – the F star of Y is convex in Y, completely independent of whether X is convex or not, I'm sorry, F is convex or not, totally independent.

And it's really stupid; look at this, what king of function is what's uncovered there above my hand and to the left – right of my finger as a function of Y? It's affine – that's affine in Y for each X. Hey, that's a supremum over affine functions. That's convex, period.

So this function is convex, okay? Actually I can tell you something fun – I'll go ahead and let you – no, I'll mention it now, its fun. Now you know usually – you've seen conjugant's right? You've seen conjugate's for complex numbers, you've probably seen conjugate's in some other – I mean as a bunch of places work.

But, when someone uses in a mathematical context the word conjugant, and there's some sort of cultural background there. A conjugate is usually something which when applied twice does what?

I mean we're just talking roughly, right? If you do a conjugate twice, what do you get?

Student: The original.

Instructor (Stephen Boyd):Part of the original thing, or you know, something – and if you've seen this in a lot of different – other context, it might be something related to it like the closure or something like that, that would not be surprising, the conjugate.

So you might guess that the conjugate of a conjugate of a function would be an interesting thing, and you might even guess it was the function, but I'll tell you what it is and its super cool. This is just for fun. I'll tell you what it is; there's a function, and the conjugate of the conjugate is not the function, it's something called the convex envelope of the function.

And it is the – let's see if I can say it right – it's the largest convex function that fit's under this one. I think I said that right. So in this case here's what you do. In fact, I can describe them to you as epigraphs very easily, ready? It's this; we have to have a name for the convex envelope, so F enth is going to be the convex envel – is going to be the function which is the convex envelope of F, and I think I can write it – I hop this is gonna work, there we go, it did work.

I don't know if that's clear. This says that you take the epigraph of a function and you form the convex hull of it and that's the epigraph of the envelope. So the – let's sketch here the convex envelope of this function. Well it's gonna look like this, it's the function itself out here.

It turns the corner and then it goes like this – ignore that, my little – and then it hugs the function again. Make sense? So that is what just if you're curious, this is. It'll come up, you'll see, later. So that's what it is. By the way, for a convex function, roughly speaking F star start is F.

Roughly speaking, means that there are some technical conditions and things like that. Okay. Let's look at some examples of the conjugate, and this is actually a calculation you're going to have to do a bunch of times. It's going to come up in duality and things like that, so you're gonna have to do conjugate calculations.

So I'll just do one and then I quit, and then you'll do some, I promise, and then that'll be enough. Then after that you'll do it on demand when you need to.

So the negative log rhythm is convex, let's calculate it's conjugate. Of course I can calculate the conjugate of the log rhythm if I want, I can [inaudible] of anything I like, but we're interest mostly in conjugates of convex functions.

So the – lets take a look at this function, so this looks like this, I guess that's the negative log, something like that. That's the negative log, and I want to work out the sup or maximum over X of XY plug log X. Now log X of course is unbounded above. So obviously here if Y is 0 or positive, this thing is unbounded above. So the supremum is plus infinity, okay? So that much is clear.

Now let's assume then that Y is negative. This is interesting, because now it's a log that's growing, but then you subtract a linear thing and in fact a linear thing will win out on a log always, and they will – to find out exactly where they – it reaches it [inaudible], you just take the derivative and it's a – these are smooth things.

So you take the derivative of XY [inaudible] X and you get Y plus 1 over X is equal to 0. That is the point X that would maximize XY plus log X, provided Y is negative, okay? So this says that X is one over minus Y, okay? And that's a – and then you plug this X back into XY plus log X and you get minus 1, that's the XY part, plus log 1 over minus Y, okay? And that's this thing, that's the conjugate, and that's for Y.

And it's important when you work out a conjugate to get the domain correctly here, so okay. If you have a strictly convex quadratic, so that's one half X transpose QX – by the way, the one half is here because – the one half is there often because it simplifies some formulas or makes them look prettier.

Other cases where you're not like basically calculating gradients, you leave the one-half off, and – it's sorta you choice. Actually, this is one of those forea transform like things where you can stick the one over two pi anywhere you want, and of course everybody does it differently which is pain in the ass, so. So, it's one of those stories, but if you put the one half here you're gonna get something pretty.

So if you work out the supremum of Y trans – this is a quadratic function [inaudible] equal to 0, you get X and all that, and you get a beautiful thing that the quadratic, the conjugate of a quadratic function associated with quadratic – with positive definite matrix Q is – is the quadratic for with positive definitive matrix Q inverse.

So that's what that comes up in. Conjugate by the way has all sorts of interpretations, but I put them off until – like in economics and things like that. I put them off until we get to duality, and we get to cover everything on this.

Quasiconvex functions so I think we have only one more of these and then we're done – you done with your first dose of convex functions. Oh, I should let you know there's like a whole industry, in fact almost like a little subculture of – it's been going on for like 40 years, where people worked out little variations on convexities, so there's quasiconvexity, there's sudoconvexity, and I forget, that goes on and on and they – anyway, and – it keeps otherwise dangerous people off the streets so it's – it provides a positive – the net benefit to society I believe is positive.

And it's fine or harmless people, they're nice enough, but just to warn you, you would find tons and tons – and they get of course all bent out of shape and they make little diagrams with – this show what implies what, and they go, did you know that a strictly quasiconvex, blah, blah, blah, is actually pseudonormal convex? Anyway, and they'll make – draw diagrams.

And if you don't adequately say, wow, that's really interesting, and then you'll be in trouble. So that's the best thing to do, that's my recommendation when you are confronted with someone. By the way, someone who pesters you and says did you quasiconvex or sudoquasinormal strictly convex.

I don't even know what they do; you can look on Google to find out all the different things. We have looked at all of these things, the 27 different extensions, and identified two that seemed above threshold. So I just want to let you – this is one of them, quasiconvexity.

So quasiconvex is this; its s function whose sublevel sets are convex. Now of course any convex function satisfies that, but that's not enough and here's an example which is obviously not convex, but every sublevel set is, so want happens is here, to get a sublevel set you just put a threshold. You know, I really should put it above like that, because you want to look below, I don't know.

You need a fishing line to put across here. And you pull it across here, and you can see for example, the alpha sublevel set is an interval the beta sublevel set – I mean assuming that this continued indefinitely, is actually an half infinite interval, but it's convex, so this is – this is fine.

And you call something quasiconcave if it's – if the super level sets are all convex. So let's just look at some examples to see how this works. Squared the absolute value of X, I guess that's a function that looks like that; that's quasiconvex on R, and it's obviously not convex.

It's quasiconvex because you – you just simply put a line anywhere and you get an interval. Here's an interesting on, the ceiling of – of a number. So the ceiling of a number is the – it's the smallest integer that's bigger than the number – bigger than or equal to the number, that's the ceiling.

Now this is pretty weird, because that's integer valued, okay? Now integer valued functions, for example – well let's have a discussion about integer valued convex functions. Who would like to start the discussion? What can you say?

Let's discuss the theory of integer valued convex functions. Do they exist?

Student:[Inaudible]

Instructor (Stephen Boyd):Constant, okay. For example here is a function, 3 convex, let's keep going, anything else?

Student:[Inaudible]

Instructor (Stephen Boyd):Four, thank you. It's going to be a long discussion at this rate. Okay. What do you think? I'm not asking – I don't want to prove – we're talking eyeballs here.

Student:[Inaudible]

Instructor (Stephen Boyd):Yeah, okay, so it can't – so the only – these are the only ones, right, the only integer valued – you couldn't have two values, like you couldn't have 3 at one place and 4 at another, okay. So that was a very – you are now in total command of the extensive theory of integer valued convex functions. Basically there hardly are any other than – they're known as constants, actually.

So that's why this is a bit interesting, that the ceiling is gonna – which is obviously integer valued, and it's going to be quasiconvex. And actually, what's kinda cool about it is we will be able to optimize things involving these. You will, I absolutely promise, there'll be problems involving like FIR filters, again, if you're in EE, and we'll ask questions like, what's the minimum order required to have one decibel ripple over this [inaudible] minimum 40 decibels rejection over here.

Now when you just hear that question the convex part of your brain is not lighting up, because you just hear – you heard order of a filter, that's an integer. Being familiar with the theory of integer valued convex functions and knowing that it's a very short theory, you might say, well there's no way you can optimize something involving an integer, well you can't, so we'll get to this.

That's quasi [inaudible] and it's easy enough to check. These are just stupid things, every ceiling, the sublevel set of a ceiling function is just an interval, and so it's very easy. Log is quasilinear – product X1 X2, that's a quadratic form that is neither convex nor concave. It's the quadratic form associated with that matrix.

That matrix, just take a look at it, it's not positive definite and it's not negative definite, it's got split Eigen values, so it's neither. It's certainly neither.

Student:[Inaudible]

Instructor (Stephen Boyd):Quasilinear it means quasiconvex and quasiconcave. And by the way, these functions do not have to be linear, not even remotely. A linear fractional function is quasilinear, meaning it both quasiconvex and – this is the most famous by the way, because this function here is absolutely not convex.

If you have a liner fractional function it looks something like this – well there might be a place where it goes into minus infinity or something like that, but it's gonna look like this – like that. I drew the wrong one; it doesn't have to look like that. All right, I take that back, it can be, depending on A B C and D, sorry. But in general it doesn't matter.

A liner fractional function is gonna be quasiconvex and quasiconcave. There are famous problems that have this form. Here's an interesting on; distance ratio, so the ratio of your distance to two points, on the set of points where you're closer to one than the other. This is a half space by the way, which I think – it's a half space. So as long as you're closer to one thing than another, then your ratio of your distance to that then the other is actually gonna be quasiconvex.

But the way, all you need to do is take this function and put a gamma – take it to the gamma, like if you're in wireless and you have here the SIR from a good transmitter and a bad one. And it says, -- well you have the inverse SIR, and it says that the inverse SIR is convex, quasiconvex, provided you're closer to the transmitter you want to hear than to the interferer.

Sorry, I have to throw these in, because I'm bored. Sorry, go on?

Student: Is it the same of convexity preserving operations that work on these quasiconvex [inaudible]?

Instructor (Stephen Boyd):No, that's what cool, it's not. It's a more general set. For example, a monotone function of a quasiconvex function is quasiconvex, because quasiconvex just basically says, you know, all that matters is your sublevel sets. So if I put some monotone transformation it works just as well. So it's a different set.

Oh, and it's neither a restriction nor a generalization of the others. For example, the sum of two quasiconvex functions is generally not quasiconvex. And let's just look at a quick example to show that; here's a quasiconvex function, here's another one, and the sum, for sure, is not.

Oh, there's another name for quasiconvex, it's called unimodel, that's another name. What reminded me was I added these two together in my head and that's what we would call not unimodel.

Okay, so let's look at one that's not nontrivial, just for fun, it's the internal rate of return. So here I have a cash flow sequence, so X0 up to XN and then XI is a payment in period I, and positive means to us.

So, we'll assume that X0 is negative, that means that we pay out at period T=0 and we'll assume that the sum of all of these things is positive, so I guess that's what we'd call a good investment. The sum of – if these are positive it means it's a payment we receive, and this is the sum total of everything and it's balanced out by our investments.

So minus X0 is the investment. Then if I work out the present value of this cash flow stream, with an interest rate R then that's nothing more than this, I sum from I =0 to N XI with 1+R to the minus I.

So that's my present value assuming an interest rate of R. Now the internal rate of return is defined – usually completely, incorrectly, and ambiguously in – in fact essentially every text I look on in finance. It actually said it's the interest rate that makes the present value 0. Well let's see, I would say somewhere in middle school you would learn that – for example, a function like that could actually have a 0 at – there are many values of R for which it can be 0, but this apparently hasn't stopped people from defining it as V.

So the correct one is it's the smallest interest rate which makes the present value 0, so that's the definition of the internal rate of return. For example, if your rate of return is 8% means that your cash flow stream has a present – or let's say the income derived from the cash flow stream is exactly the same as had you invested it at an interest rate of 6% or 8% or whatever I said. So that's the idea.

Now this is a rather complicated function of X, so in fact even – I guess not of a particular one, but if you had a whole bunch of different cash flows and things like that, it's quite complicated to visualize what – and the way you visualize what the internal rate of return of a cash flow stream is – well you really have to just form this sum.

That's a function of R; note it's a function of powers of 1 over 1 plus R, so it's actually a polynomial in 1 over 1 plus R. Then you'd find the roots of the polynomial. These would be the values of the interest rate at which that investment is neutral and then you'd find the smallest of those.

This is complicated; it's like an Eigen value or something. This is not simple. It's quasiconcave. By the way, this is your first hint that something cool is happening, I'm going to mention it; because this is the first time we ever hit it.

We're going to find out things like quasiconcave functions, we're going to be able maximize. You'll be able to maximize this, five lines of code with complicated constraints, so you'll be able to maximize this. We can minimize convex functions and quasiconvex functions, we can maximize quasiconcave ones.

Now it turns out the – it's hard to minimize internal rate of return, that's a very difficult problem. What this says is it's actually easy at least algorithmically and mathematically to maximize internal rate of return.

Now if you think about it carefully in practice, we're not neutral. Problems of maximizing and minimizing internal rate of return do not occur in equal – with equal frequencies, right? Actually pretty much you never want to minimize internal rate. You want to maximize internal rate of return, and you'll wow, look at that, the thing we want to do turns out to be the thing we can do.

I just thought I'd mention this, because it comes out – I mean it's kinda stupid, you don't mention this in like 263, you say, isn't that amazing, we can minimize norm AX minus B. Every now and then we want to maximize it. Just though I'd mention it.

So what is – how do you show this? To show something as quasiconcave you look at the super level sets. By the way, what you're doing is you're saying please let's study the set of cash flow streams that have an internal rate of return exceeding 8%. That's what you – that's a rather complicated set.

That's like the set of symmetric matrices whose Eigen values are less than – whose maximize Eigen is less than 7. I mean it's not impossible to imagine what it is, but it's quite complicated. What is it? To say that your internal rate of return is bigger than R, that means that for all interest rates up to R your internal rate of return – I'm sorry, your present value is actually positive.

So, that's this. That's an infinite set of inequalities, one for each interest rate up to capital R. I'll let you check this, but it's a quick way to see this. By the way, if you tried to do this in cases of a sequence of 2 or 3 using the formula for the roots of a polynomial or something, this would not be fun.

Now for quasiconvex functions there's a modified Jensen's inequality, and let me just draw it, because it's better to draw it than to – than – I don't know what's here, so. Jensen's inequality says something like this; you have two points like this, and let me just draw a convex function first.

And it basically says that if you draw this line, this cord lies above the graph. That's what Jensen's inequality says. It's says actually you do a better job – for example F at the midpoint is smaller than the midpoint in the average of F. That's basically what it says, that's convex.

Quasiconvex is cool. Let me draw – there's a nice quasiconvex function, and I'll pick some points. I'll pick this one and I'll pick that one. Oh, no – well the cord lies above it – it doesn't matter. I mean, in this case the cord would lie above it. But I'll draw the picture of what you have to have for a quasiconvex function.

Instead of the core, what you do is you draw this thing at the maximum of the two values. So it says the function has to lie below there, which it does. Another way to say it is if you have a quasiconvex function and you have one X and you have another Y, and you for a linear convex combination of X and Y, and you evaluate F, it's not worse than the worst of X and Y at the end points. That's a way to say it, if worse means large in F. Does that make sense?

So this is the picture, and the way you write that is this way, that along – when you form a blend and evaluate F, you're not worse than the – you couldn't be worse than F at the end points. That's the picture.

And you have some things like this, if F is differentiable, if it's quasiconvex – this occurs if there's a characterization here, and it basically says this, that if the – it says that the sublevel set of F lies on one side of the gradient picture. So if you take the gradient like this and form a half space, it says everything over here has a higher or equal value of F, that's what it says.

Or another way to say it is every – and for a quasiconvex function, every point that's better lies on this side of this thing, period.

Okay, that's another way to say it. By the way that's a hint that we can minimize it, because when you evaluate a gradient and it gives you half space and it says, I don't know where the minimum is, but it's got to be in this half space.

You can thing of that vaguely as query – by evaluating a gradient you got – this is very vague and weird, but it basically says you got one bit of information, because it basically said it's in that half space, not that one.

We'll get to that later, but that's why this would work. And finally, in our whirlwind tour, the second of the two extensions and variations that were above threshold. This one is very simple, but it's important enough.

It's not like it's an extension or anything, it just happens a lot. And it comes up – it's log concavity. This is weird, because it turn out – to say something's log concave, just says log F is concave, that's all it says. And it's a weird thing, it says that if you evaluate the function at a point on the line say between X and Y, it's bigger than or equal to the weighted geometric mean.

It's the same as if you just take logs here, you get this. Now for some reason applications come up overwhelming log concave, not log convex. I don't know why. It doesn't matter, I'm just telling you. Log concave things come up all the time. Log convex things like never come or almost never come up.

It could be that in all the applications where log – where this thing comes up – which is statistics, it's everywhere in statistics. We'll see it's in lots of things. But everywhere where it comes up their conventions are that they want to maximize things, like you want to maximize likelihood rations and maximize posterior probabilities and things like that.

So here's some examples; powers are log convex for positive exponents and log concave for negative ones. And this is very, very important, many, lots and lots of probability densities are log concave, okay? That's very important. And that's called a log concave probability distribution.

An obvious example would be a Galician, because if you take the log of this thing, and let's look as a function of X, you get a negative quadratic.

Another one would be like an exponential variable or a Laplasian variable with a two sided exponential distribution. But actually tons and tons of distributions are. So if you're in statistics or something like that, you know all – various families, all the ones you can probably think of right off the bat would be log concave.

Things like beta distribute – pretty much everything. Pi square, you know all of these things are gonna be log concave. The log concavity is going to be the basis of a whole bunch of bounds in statistics, so if you're in statistics this comes up and it may even – it might even have been mentioned explicitly, although I doubt it.

Actually who here – I know there's people in statistics here, they're not gonna self identify though. I'll find out. I can find out who's in – anyway. Has log concavity ever been mentioned in the statistics – yes, cool, what class?

Student:[Inaudible]

Instructor (Stephen Boyd):In seminars, oh, okay. Not in like – how about 310 – no, it wouldn't be in 310, it would be in the statistics stuff.

Student: In 300 it's mentioned in passing.

Instructor (Stephen Boyd):Cool, good, okay, there we go. Here's one, totally non-obvious, has amazing applications like this immediately. Is the cumulative distribution of a Galician. So it's got different names, I think urpsey, I think in ee and communications they decided it wasn't good enough [inaudible] and the called it the Q-function or something, is that correct? Something like – isn't that what that is?

Student:[Inaudible]

Instructor (Stephen Boyd):One minus – it's what?

Student:[Inaudible]

Instructor (Stephen Boyd):One minus five, okay great, that's good for my colleagues to introduce a new notation for something that's been used for hundreds of years by many other people. Sorry, let's move on.

So the log of accumulative distribution, and I suppose the log of a q-function is long concave, okay. And you know, this is not so obvious. The cumulative distribution looks like this of a Galician, it goes like that and then goes up like that. It looks like that. If you take – it's obviously not concave.

You take the log, and the part that's 1 – well that's 0 obviously [inaudible] and this part curves down like that. And it's asymptotic in fact to something like a square here, which is the asymptotic of the Galician, the tail of a – small values here are well approximated by Galician itself.

Student:[Inaudible]

Instructor (Stephen Boyd):No, that's false. I'll tell you very shortly which are.

Student:[Inaudible]

Instructor (Stephen Boyd):All which?

Student:[Inaudible]

Instructor (Stephen Boyd):In one [Inaudible] that's true, because they're monetel. So they're definitely quasiconvex. We'll get to this. So let's look at some properties of these things. Well, just by working out the hessian of the log of a function is, it's something like that [inaudible] or something.

You get this, it says basic – it's actually quite interesting, we can look at it, it says that F of X – if you're talking about a log concave or a log convex function, obviously the function itself has to be positive, or non-negative at least. It can be 0 by the way, which in log will map to minus infinity which is perfectly okay, because for a concave function minus infinity is essentially our OOD token.

I gave it another name the other day that was out of domain. What did I call it before, NID. Anyway, one of those tokens – it's the token for being out of the domain, so minus infinity. So this is non-negative, and this is actually quite interesting. I guess you could put the F of X over here or something like that.

This says to be concave is to say that the hessian is less than or equal to 0. This says actually the hessian can be less than or equal to. That's a rank 1 – that's a dyad. So this actually says, you're actually allowed one positive Eigen value in the hessian of a log concave function.

That's cool. One positive Eigen value you're allowed. So it means you have one direction along with your curvature can actually be positive. We needed that because look at this, okay. But it's got to be overwhelmed by – it's gotta be no more than something having to do with the gradient here.

I mean I don't want to give too many interpretations of this, but. Here are some obvious things; product of log concave functions is log concave, totally obvious, because the product of two things, you take the log it's the sum. So that's the sum of convex functions.

Sum of log concave functions is not always log concave, that's easy. Add to make a mixture of Galician, moved apart you get tow bumps. Oh, by the way, if it's log concave its quasiconcave, because log is monotone. And you can't have two bumps on a quasiconcave function, because then a super level set will be two isolated intervals so it won't be convex, the set.

Now we get to something which is highly, highly, highly non-obvious, it's this; if I have a function of two variables and it is log concave and I integrate over one of the variables, the result is log concave, it's true.

For literally years I have looked for a simple proof of this, and simple means like simple, assume everything you want, X and Y can be one variable, F can be differentiable as many times as you like, so I'm talking about a simple proof that would please our careful mathematician friends, I'm talking for just a normal person where you just write out the first and second derivatives, look at some stuff and say – and you end up with this and say, there it is.

I'm just talking about something, a bunch of calculations that would establish it. I have failed, always. By the way, people – I've said this challenge out to a bunch of people, and several time people have come up with – by the way, I have a four page one which is very complicated and anyways.

So if anyone every finds one out – figures out a simple one for normal people that just uses calculus or something like that and a couple of arguments here and there, it'd be great. After which I will say that it was obvious, but at the moment it's obvious but unknown to us, this simple proof.

It's certainly not known to me, that's for sure. It's not obvious. The implications though are stunning, and I will show you some of them. Convulsion preserves long concavity. That gives you a lot of stuff right there. So if you have – by the way, convulsion it means if you have two random variables each with log concave density and you add them, the distribution of the sum obviously is gonna be log concave, so that's the kind of thing you would get just right off the bat.

So convulsion preserves log concavity and here's a fascinating app – is this, it says if you have a convex set and you have a random variable with log concave PDF, then the probability – this is a function of X, that X plus Y is [inaudible] is log concave.

And the way to show this is I just apply this one integral. Your integral thing is – that's the big hammer, you just reduce everything to the one as of now, non-obvious fact that you can integrate over one set of variables a log concave function and the result is log concave.

So roughly speaking, integration preserves log concavity. And you apply that – you'd make a function here, G of X plus Y, which is just the indicator function. It's 1 of you're in the set and 0 if you're outside and work out this integral and it's that. And this is very cool, this immediately has amazing applications which I will show you one right now.

So here's the way it works; I have some set of values like this in some multi-dimensional space like that, and these are – you should think of these as acceptable values of what happens. So these – each axis could be – I don't – it doesn't really matter. This could be the – I'm setting the process recipe or the target process recipe or whatever for making IC's.

One of these could be threshold voltage which if it's to high the chip is too slow. If it's too low it dissipates too much power. By the way, how many people are in circuit design? Now come on. Does anyone care about it? Maybe they just sleep late. They were in the clean room until very late last night and – I know they're in this class.

But if no one is here then I will remove these examples from the – all right. So this is the acceptable [inaudible] and what's gonna happen is you're gonna set a target – you're gonna set the target value like that and you'll adjust all your manufacturing to hit that target value.

By the way, when you do manufacturing runs and you look what comes out on Monday, Tuesday, Wednesday, Thursday, and things are biased away from the point – if the mean of that is away from that point you need to get new people running the manufacturing, because you just – you would offset it.

So we can assume that when you hit the target point the distribution of values [inaudible] will come off in manufacturing will have that mean. I mean if it doesn't have that mean, then you shift it or whatever.

So you have a mean, and let's say that the manufacturings [inaudible] are given by this random variable W. It could be some horrible thing, some joint distribution, but we assume the following, its log concave PDF.

And Galician would work just instantly, but that's it. What's gonna happen now is this. It means that when you hit a target here, you're gonna get values around like that or whatever, every now and then you'll be out. If you're out then that's something that doesn't perform. That's something that's no good and that's a waste.

And your yield is the probability that you actually end up in the set. This all makes perfect sense. Now, imagine taking the target and moving it around. As you move it around your yield changes, obviously, right.

So for example, if you put the target here, what would the yield be? I'm not endorsing that as a choice by the way, but what would the yield be here? Would it be 0? No, probably not, because every now and then accidently you would manufacture an acceptable part.

I mean if the distribution doesn't have fine eyed support, so every now and then accidently. And this would be a poor choice, and it's totally obvious what a good choice is, you want something deep in the set, and that's got all sorts of names.

Its called design centering, is one of the names for it, and yield optimization, it's got all sorts of other stuff. Although, yield optimization means something different in other contexts, so. Everybody got this? And now you should also be able to visualize also the yield as a function of the target point.

So it's high here, low here, less low here, and high is somewhere –at some point in the middle here. So, here's the point. That function is log concave. Just very roughly speaking, let me tell you what that means. It means we can optimize it. We can maximize it. So you put these two together, and what it says is you can actually do – you can actually directly maximize the yield.

Can you make a note? We'll make a homework problem on that. Not for homework 2, later, don't worry. We'll do a yield maximization problem. Okay, so that it. One last topic on convex functions is convexity with respect to generalized inequalities. Generalized inequalities allow you to overload ordinary inequality.

All you need to define convexity of functions is you need an id – you need the ability to talk about – well in the very general case you need to talk about the line segment between two points. We work in vector spaces, so for us that's fine. And then you need this inequality to be able to say that like F of theta X plus 1 minus theta Y is less or equal to theta [inaudible].

And so you need that inequality. These can be vector valued and then you can talk about convexity with respect to a generalized inequality. The only case where this would really come up and is actually anything is with matrix convexity. And so there it just says the inequality holds, but the inequality has be interpreted as in terms of matrices.

So – now I think we'll quit here. This sort of finishes our whirlwind tour through I guess the [inaudible] of convex sets and functions. Next time half the lecture will be a little bit boring, because it's just nomenclature of optimization, but by next week actually I think things will get interesting, so we'll quit here.

[End of Audio]

Duration: 77 minutes

ConvexOptimizationI-Lecture05

Instructor (Stephen Boyd):They should install it before then. Okay. Great. We'll start [cuts out] announcements, the first which we put on the website also, which is the authoritative source of information is that this week's section which would have been yesterday will in fact be today. As a matter of fact, it'll be at pretty much the worst possible time. It'll be at 11:00 a.m. today, in other words almost immediately following this lecture. So it'll be 11:00, 191 Skilling, so I guess that's upstairs. That's the section this week. Let's see another announcement is just to remind you if you were a victim of EE Quals and have filled out the proper paperwork, then your Homework 1 is due today. Then you should jump right into Homework 2. And we're working on Homework 3 for you, so it should be – Homework 3'll be nice.

Okay. I'm trying to think of this. Any other announcements? Oh, there's one other. In the next week, we're making the transition in the class – actually, even in today's lecture from sort of boring background material to actually stuff that's useful. So that's actually gonna happen in the next week. But part of that is that by next Monday and by Homework 3, there'll actually be a software component to the class. So next Monday's section – I'll announce this again, but next Monday's section will be on CVX, so we'll explain how that works and so on. It's actually quite well documented. I would recommend if you get bored or something like that, if Homework 2 isn't quite enough for you, or something like that, you get bored or who knows what, what you should do is install CVX. Now actually you should install it I'd say periodically as we silently fix errors, and you can watch the build number increment. But you should go ahead and install it, maybe read the manual. This is just if you get in some weird mood where you feel like doing this, you should do that. Anyway, you won't have a choice. By next Monday, you will have to install it and start reading the manual and all that kind of stuff. And it'll tie into everything you've seen so that should be okay. So that's not really – it's just looking ahead. The other of course is I should mention that the reading assignment is that you should be reading Chapter 4. Actually, hopefully you should have skimmed it already at the very least because that's what we're gonna work on today, and probably the next lecture as well. So any questions about last time? Otherwise, we'll zoom forward and actually get to – actually, it's gonna almost be useful soon, maybe even today.

Student: Are you gonna discuss generalized inequalities?

Instructor (Stephen Boyd):Am I gonna discuss generalized inequalities? No. That was the question. That was my answer, too. No. No, it's clear enough in the book. And when we get to something where it's relevant, like experiment design – it'll also be relevant in detection and estimation, then I'll go back over it. Also in multi-criterion optimization which we're gonna do later. So I'll go back over it. Okay. So today we're gonna go through optimization problems. The first part is a bit boring. It's just setting down basic terminology like what does it mean to be feasible, what are the constraint sets, actually what is a convex optimization problem, but then it'll transition to actually useful, so you actually find out about what a linear program is, what a quadratic program is, second order cone program, and these types of things. We'll get to those maybe later – hopefully, this lecture. So we'll start with just the boring nomenclature of an optimization problem. So an optimization problem looks like this. This is a notation for it is to minimize an objective, f0(x) subject to fi(x)=0. That's from i = 1,…,m, and hi(x) = 0. So this is sort of a standard form for an optimization problem.

By the way, if you read the Russian, the Soviet literature, you'd see different things, which is actually – I guess many Russians are now teaching in U.S. and European universities, and it's now propagated and you might as well know what it looks like. They would write it this way. It's actually a perfectly good notation like that. So you will see this, and it means that f0(x), the objective is to be minimized. So I guess I don't know how to pronounce this in Russian, but it's something like that. So you'll see this, and you'll see some other ones. Now you'll also see some things that drive me insane, and I'll mention one of them. You'll see this, and that really drives me insane because the semantics is wrong. Min actually has a meaning. Like you have the min – min takes as argument a set of numbers. In the strict definition, min takes as argument a finite of set of numbers, and it returns a number. Minimize is actually a verb. It's not a mathematical operator, and it's part of the verb that you use to create an optimization problem, so these are different things. But you'll see this.

Actually, sometimes they just barely escape by putting a period at the end, which suggests that it wasn't min. It was short for minimize. But anyway, you'll see that. Sloppy. Now here x is the optimization variable. This is called the cost. It's sometimes called the objective. It's got all sorts of other names or – if it's in a specific application area, it'll have a name, too. Then these are called the inequality constraint functions, and the hs are the equality constraint functions. Now here the right hand sides are zero, but we're gonna see very quickly how you would – if the right hand side weren't zero, how you could transform it. So it's just useful to have a standard form like this.

Now the optimal value of this problem – so the type of object or data what this is that's an optimization problem without the min, or with the min period. That's a problem. By the way, it has no numeric – by itself, you can't say this problem is three or something like that. But you can talk about the optimal value of a problem, and that's simply the infimum or minimum of f0(x) over all x that satisfy the equality constraints. And there's a completely standard convention that the infimum of an empty set is infinity, so that's absolutely standard. That says you'd say that the optimal value of this problem is infinity. If there's no feasible x, if there's no x that satisfies these inequalities and equalities. And that makes sense because if your objective is to minimize something, then the value plus infinity basically means you've done infinitely poorly. You could not have done more poorly if your job was to minimize something if the value is infinity. And that is after all what infeasibility means. It means you didn't even get to the point where you could actually compare two points and say yours is better than mine because there weren't any that met the hard constraints. Okay. Now on the other hand, this could be minus infinity, and that would be the case if there were a sequence of xs all feasible with f0(x) k say going to minus infinity.

By the way, in different problems this is referred to different ways. For example, in something called risk sensitive estimation, they have a wonderful name for that when the problem is unbounded below. It's called euphoric breakdown. That's the technical name for that case. They have all sorts of other names for it in different fields and things like that. I guess it's infinitely good, I guess. What it really means is you probably set up – your modeling of your problem is not that great, but okay. So mentioned the idea of optimal and locally optimal points. Well, you say a point is feasible if it's in the domain of the objective and it satisfies the constraint. By the way, if it satisfies the constraints, it must also be in the domain of the constraint functions because you certainly wouldn't say it satisfies the constraints if it didn't also – to satisfy the constraints, fi(x) actually has to return a number and not the NID token, the not in domain token because there's no way, if that returned not in domain, that you could actually say that the inequality was valid. Okay? Okay. And a point is optimal if it's feasible and if its objective value is equal to the optimal value of the problem. That's optimal. Capital Xopt not totally standard notation, but I think fairly standard. That's the set of optimal points. And this can be empty, for example if the problem is infeasible of course, it's empty. If the problem is unboundable below, this is empty. And it can be even when p* is finite, this can be empty. So there's – you can have everything, in which case –

By the way, to say that Xopt is the empty set, the way people would say that is you'd say the optimal value is not achieved is what you'd say. If it's finite, it – I'll give you an example. Let's just minimize 1/x on the positive axis. So if you minimize 1/x, the optimal value is obviously zero, but it's not achieved because there's no x for which 1/x – there's no x in R++ for which 1/x is equal to zero. So you can get as close as you like and so on and so forth. So in this case, you would have Xopt is empty, and you would say that's a problem with optimal value zero which is not achieved. Now you say a problem is locally optimal if when you add a constraint – notice that this depends on x – if you add a constraint, that you not stray far in norm from x. R has to be positive here. If you can add such a constraint, and x becomes optimal, then it's locally optimal. And then these are kind of – you can just work through these. These are very straightforward. So if you minimize minus log, it's unbounded below obviously because I just take infinitely large xs and the sequence of xs getting larger and larger, negative log gets smaller and smaller.

Entropy here – that negative entropy – that actually has a perfectly well-defined minimum that occurs at 1/e. You just differentiate, set it equal to zero, and the optimal value is -1/e. So this would be one with none of the pathologies here. If you have f0(x) = x3 – 3x, here it's unbounded below. Obviously, as x goes to minus infinity, this thing goes to minus infinity, so it's unbounded below, but you have a local optimum at x = 1. So if you take a look at this, you will find that around x = 1, this thing goes like that. The derivative is zero, and in a little neighborhood around it, actually it's locally convex there, and that means as you stray from x = 1, the objective value goes up, so that's locally optimal. I'm going quickly over this because it's kinda boring. And then when we get to the actual specific problem classes, it's more interesting. Now when you have this standard form problem, when you write this down, there's the implicit condition – this basically says this function is what you're gonna use to compare to proposed points that meet the basic requirements. The basic requirements are that you should be feasible, but there's even something even more basic.

You have to be in the domain of all the fs, the objective and the constraints, and you have to be in the domain of all of the constraint and objective functions because if you're not in the domain of them, it means that one of these things evaluates to – it doesn't make sense. And you would certainly not say something is feasible if one of these things – if you were out of the domain of one of these. So there's an implicit constraint that the only xs you'll consider are in the domain of all functions, all objective and constraint functions, the objective and all constraints. So that's sometimes called an implicit constraint because you don't write it down explicitly. Actually, merely by writing down the symbols fi and then called on x, you are actually imposing the constraint that x is in the domain of fi. So that's why it's implicit. Later we'll see there's actually a difference between implicit and explicit, and it will make a difference, but it's something to think about. When you write out the constraints explicitly like this, these are called explicit constraints. And you say a problem is unconstrained if it has no explicit constraints. Here would be a very common example, one in fact that we'll see a great deal of. It's minimize the following function. It's the sum of the negative log(bi – aiTx). Now to talk about the log of something, at least if you're not in a complex variables course, not in the context of a complex variables course, to talk about the log of something, this has to be positive.

And in fact, our definition of log which would differ of course from the definition of log used in a complex variables course, complex analysis course – our definition of log has a domain that is R++. So merely by stating log – if I write down log(u), I don't even have to put that in a constraint. That has a side effect, and the side effect – I'm talking about a side effect the way when you call a function. That has a side effect, and the side effect is that requires you to be positive, so that's the way it will work – I mean this is all standard, but that's the idea. So here, these have to be positive here. Now to say that bi – aiTx is positive says that there's actually a positive slack in the inequalities aiTx < bi like that. So the set of x that satisfies this is actually an open polyhedron. So it's an open polyhedron where this even makes sense, the domain of this – open polyhedron. Now on that open polyhedron, this function here is convex, smooth, everything. So what you would say is if someone asks you, "What about this problem," you'd say, "Well, that problem is unconstrained." I guess maybe the right – if you really wanted to be – specify, you'd say it's formally unconstrained. It has no constraint functions. That's what you'd say.

Of course, there's an implicit – and then if someone says, "Unconstrained? Does that mean I can throw in any x I like?" The answer to that is no. You can throw in any x that satisfies this, and that's an implicit constraint inherited from the domain of f0. Okay. Let's see. Let's talk about the feasibility problem. One variation on the optimization problem is this, and it's sometimes written this way. Find x subject to – by the way, you will also sometimes see subject to written as such that, which I don't actively – That's a question of style, and by the way, this is bad style. But nevertheless, it's within the – it's only in the – it's between the green and – it's somewhere in the yellow range for me. It's not – that's not what such that means in English, by the way. Subject to are the English words that say exactly what happens when you do something subject to restrictions. You don't do something such that restrictions. That's actually not English, but you will see this, but not from me you won't. And the worst of course is min this. Oh, and by the way the initials S period T period go either way, though it's really your – I don't know, so S period T – that's even worse, though. So find x subject to these constraints.

This just says there is no objective, and what that means is that any x that satisfies these constraints is equally attractive. There's just nothing – you're completely neutral. It just means find anything that works there. That's called a feasibility problem, and it's really the same as this, or it's convenient to make it the same as this. You make it minimize the constant objective function zero subject to, and then these inequality constraints. Now that looks stupid, but it's not and it encodes everything correctly. So what it means is this: the optimal value of this problem is either zero or plus infinity. If there is a feasible point, then – any feasible point here is optimal because if I have a feasible point, I could ask what is its objective value, and the answer would be zero, but that's as small as the objective value gets among feasible points. If there is one, that's p*. Therefore, any feasible point is optimal here. On the other hand, if it's infeasible then the p* is you take the infimum of zero over the empty set, and that's plus infinity so everything works out just fine when you do this. Yeah?

Student:[Inaudible] x offset, just the intersection of every domain and every constraint?

Instructor (Stephen Boyd):No. It's not the intersection of domains. The optimal set here coincides with the feasible set, so basically in this problem, in this formulation if you're feasible, you're optimal.

Student: But doesn't that include being in the domain of everything?

Instructor (Stephen Boyd):No.

Student: How can it be in a domain and not satisfy that?

Instructor (Stephen Boyd):Here, I'll make an example. Ready? Let's do one. I could say log(1 + x) is – doesn't really matter. I'll make it easy on myself there, something like that. Now I have to figure out what that means, but anyway – and then I'm gonna add the following constraint. Or I guess – let's see we haven't done that. We haven't turned things around yet, so I'm gonna write it like this. And then I'm gonna write another one which is – x = – 10.

Now to say that you're in the domain here, there's no domain restriction from this one. There is a domain restriction here, and it's that 1 + x should be positive. But that's not the same as the feasible set. The feasible set – this says that x should be bigger than or equal to ten. That's the feasible set so that they're not the same. You have to go back and look at it carefully. Now I can say what a convex optimization problem is a mere three weeks into the class. Actually, I kinda said earlier. So here it is. There's a restriction, and the restriction is this: the objective must be convex function and the inequality constraints must also be convex. The equality constraints, that is the his, must be affine. And that means I can write them this way. I can also just write it as Ax = b in very compact notation.

Now if the objective is quasi-convex, then this is called a quasi-convex optimization problem, and you might just write it out – this is where I've just substituted for the equality constraints a single matrix vector equality constraint. Now the feasible set of a convex optimization problem is convex, but watch out because there's another definition of convex optimization problem. I guess it's what I call – there's the strict definition, and there is the looser definition. The looser definition actually makes more sense, is more appealing, and so on, and in terms of actually doing convex optimization, it's useless but let me say what it is. And it sounds right. If someone just bumps into you on the street and says, "What's a convex problem?" you would say this: it's minimizing a convex a function over a convex set. What could be more natural? And in fact, that's called the abstract of the loose definition or something like that. And in fact, it's okay but I wanna point out here that for us an attribute – so to say it's a convex optimization problem, it is not an attribute of the feasible set. It's an attribute of the problem description, period.

So there's other weird ways to write Ax = b, and they will not work with these things. For example, I could write this. I could write || Ax – b || = 0. You have to admit if || Ax – b || = 0, Ax = b. So I haven't changed anything here. That's not a convex problem in the way we'll describe it, even though it's absolutely equivalent to one with Ax = b. Okay, so for us whether or not you're a convex optimization problem is an attribute of the description of the problem and not for example the feasible set or something like that. And this will be clear in a couple of cases. Here's a simple example. Let's scan this problem for convexity and see what happens. Well, the objective – what do you think of the objective? Well, the objective is okay because it's convex and you're to minimize it. So the objective is cool. Let's pass on to the first constraint.

Well, we're already in trouble with the first constraint because this constraint here defines a convex set. The first constraint – it's nothing but x1 = 0 obviously, right? 1 + x22 is positive. I can get rid of it. However, this is f1. As I've described this problem, that's f1 and that is not a convex function, and so this problem is rejected. It is not convex. And the second one also is not convex because – that is indeed a convex function of x1 and x2, but you're not allowed to have convex function equals zero. You can have a convex function less than or equal to zero. So this – I would have to call that f2 now – this actually would've been okay. That would've been fine. That'd be a convex problem because you have a convex function here less than or equal to zero. But the point is here is you take these and you rewrite it in an equivalent way. By the way, the problem – these are not identical problems. The problems are identical only if the objective functions and constraint functions are identical. Then the two problems are identical. However, they're equivalent, and we'll use a kind of an informal idea, but nevertheless completely clear idea of what equivalent means. Equivalent means that by solving one, you can construct a solution of the other and vice versa. And construct it means not using too much work. So it's something like – I mean if you've had a course on complexity theory and computer science or something, it's something like that except we're not gonna be very formal about it. It’s something like a polynomial reduction of one problem to another. So that’s what we mean by equivalent except that we won’t get too fancy about equivalences are there. For us, it’s gonna be an informal idea. Yeah?

Student: When you say they share a solution set, do you mean both in terms of the value of x and in off the value of x?

Instructor (Stephen Boyd):Absolutely. In this case, I didn’t change the objective, so that’s why they –

Student: But you could change it?

Instructor (Stephen Boyd):I could change the objective. Right. So for us, equivalent problem means this – I think the best way to say it is that you could write a very thin wrapper code around one problem that would then solve the other problem. That’s what it means. And in this case, it would be very thin. I mean what would happen is this: to solve this problem, you would solve that one, and then return the solution. So that’s a pretty thin wrapper that goes around this. In other cases, we’ll see you might do other things. Actually, we’re gonna see lots of little equivalences, and I’ll make it very clear what’s happening. But that’s the right way to think of equivalence as we’ll use it. It means writing a wrapper around one to solve the other. That’s what it’s gonna mean. And a wrapper by the way that doesn’t do like a huge amount of work. Okay. Let’s say a little bit about local and global optima. So if you have a locally optimal point of a convex problem, it’s globally optimal. This is by the way the kind of thing that people on the street would know. This is kind of the basic thing you’d know about convex optimization would be this, you know variations on it. And it’s actually very impressive if you think carefully about it. It basically says if you have a problem, if you have an optimization problem – it says if you were to restrict your search – you have a point which is a candidate for being optimal, and it says if you restrict your search arbitrarily closely, locally – but if you do a full search in there and find that there’s actually no better point locally, you can make the stunning conclusion from having examined this R ball which is tiny – in fact, it can be as small as you like – you can make the stunning conclusion that in fact even if you were to search over everywhere, there’d be nothing better.

Although, you know after a while you get used to it that the proof of these things is like three lines or something like that, so it’s not a big deal. Still, what it’s saying is actually stunning if you think carefully about it. To say that by – well, it’s actually highly implausible when you think about it. It basically says if you do a competent search nearby and find nothing better, then in fact you don’t need to go and look anywhere else. Actually, this is the best you can do. Of course, that’s inherited from all the properties of convex functions and so on. So let’s see how you would show this. It’s actually totally straightforward. Suppose you have a point that’s locally optimal but not globally optimal. What that means is there’s a point – another point that’s optimal – in fact, you don’t even need the point to be optimal. So you just need a point that’s feasible and has the objective value better than your locally optimal point. So suppose that happens. Then it’s very simple what to do. You simply construct the line segment between your local optimum and this better but feasible point. You construct the line segment, and you start moving towards – from where you are locally optimal to this point that's better. What happens is – of course, as you move on that line, you remain feasible because x is feasible, y is feasible, the feasible set is convex. Therefore, all along that line segment you will be feasible.

Then what can you say? Well, now you have a convex function that basically is locally optimal at first, but then later actually achieves a value lower. And of course, that's impossible. So that's the idea. It's very simple to show this. And I won't go through all of these details, but that's kind of the idea. What I just described is it. Okay. Let's look at an optimality criterion. Again, it looks very simple. It's very geometric, and it's gonna generalize what you've seen before, this idea of the gradient vanishing being the condition for optimality. So here it is. A point x is optimal – this is for a convex optimization problem – if and only if it's feasible, and the following is true: it says that the gradient of the objective evaluated at the point (x)T(y – x) = 0 for all feasible y.

Now geometrically, it means this. Here's your feasible set here. These are the level curves of the objective. So this is clearly the optimal point. It's the point in the feasible set with the lowest objective value. So that's right here. You write the gradient of the – actually, you write the negative gradient of the objective function. That would be the normal to the hyperplane here, the local hyperplane of approximately constant objective like this and it says that this hyperplane here actually supports the feasible set at that point. It looks like that. That's the picture. I can actually sort of explain this. It just comes from this thing, right? It basically says the following. You know the following is true, right? f(x) plus if f is convex, this is true for all y. So if these are the level curves of your function like that, and I evaluate it say here, and the negative gradient points in that direction – it says basically if I draw this hyperplane like this, this inequality tells me something. It says that every point in this entire half-space here has a function value higher than the function value there. That's what it says in particular because if this is non-negative, then that's bigger than that. So that's the basic – the basic inequality comes from there, or the basic geometric conclusion is that when you evaluate the gradient of a convex function, and it points in this direction, it says that all the hyperplanes – I guess well the hyperplane – the half-space with that normal, all those points have worse value of f, and there's no point even looking at it because it's absolutely guaranteed.

So what this says is all feasible points live within this half-space here where the objective is worse than or equal to the value there. Well, guess what? That means that point is globally optimal. Okay? And by the way, to show the if and only if, it's very easy. I just showed one way. The other way's even easier. If this is true, then this basic inequality proves it for us. So it's quite straightforward. So this is it. By the way, this innocent little condition here – and notice that if the problem is unconstrained, let's see what this says. We'll look at these in list fashion. Let's take this simple – by the way, we're gonna come back and we're gonna look at optimality criteria in much more detail later, much more detail. But for now, this is just your first glimpse at optimality criteria for convex problems.

Student: What happens if the objective is not differentiated by that point?

Instructor (Stephen Boyd):Then this doesn't apply. Yeah. So the question is what happens if the objective is not differentiable by that point, and my response was then this doesn't apply. Okay. So let's look at an unconstrained problem. Well, let's see. What would mean to say if it's unconstrained, then this is just basically for all y in Rn, so wait a minute. If I can choose y to be anything I like in Rn, this vector here can be anything I like. What does it mean to say that I have a vector – let's call that g. Suppose I told you that gTz = 0 for all z in Rn. What's your conclusion? Yeah, g = 0. But how do you know that, by the way? What's a quick argument for that?

Student: Because if g is not zero [inaudible].

Instructor (Stephen Boyd):Okay. Yeah, sure. Here's a quick snappy one. Try that z, and you'll get your conclusion real fast because you get minus norm g2 = 0. So g has to be zero. That's the only way. So this condition which looks kinda complicated – actually, the only thing you have to remember is this picture. So you remember this picture, then there's nothing more to say. So here, the condition is that the gradient is zero. Equality constraint, that – we can also get this, and this will recover these things like Lagrange multipliers which you've probably seen earlier and you will see again. So you wanna minimize f0(x) subject to Ax = b. Well, x is optimal if and only if – I mean the condition is this: you're feasible and the gradient plus AT? = 0. Now to show that – and I'll just give a quick plausibility argument, not give the whole argument for that, but I'll show that. It says basically the following: the gradient of f0(x)T(z – x) = 0 for all z that satisfy Az = b. Okay? Something like that. Well, I can – there's many ways I can write this. Actually, this is – what's even more interesting, we also know that Ax = b, and what we see is that it means z – x is in the null space of A. That's what this says.

So this says that the gradient of f0(x) is orthogonal to everything in the null space of A, so we're gonna write that this way like so. Oh, you know what? I'm sorry. I skipped a step. Sorry. The gradient has non-negative inner product with everything in the null space of A. However, if you have a subspace – if you have a non-negative inner product with something in a subspace, then it also is non-negative when you – well, if you reverse the sign of z – x, which is still in the subspace, it's still in the subspace. Therefore, you have a non-negative inner product with a vector, and you have a non-negative inner product with its negative. That means you have to be zero. I skipped that. So that says this. Grad f0(x) is in the null space of A perp or something like that. It means it's orthogonal to the null space of A. Okay? Now that is R(AT), so that says grad f0(x) is in R(AT). That means that grad f0(x) = AT times some u, and I'll later make u – I'll mess with u to make it fit our form. But actually, I'm kinda done because I could write this as grad f0(x) – let's see how I wrote it here. Yeah. I can write plus AT(-u) = 0 for some u. And I should – I guess I [cuts out] smart, I would've put the minus, but it would've been kind of weird, so anyway. There we go. That's the condition. By the way, don't worry about these. This is just your first glimpse at this. We will spend a week, week and a half on all these things later in much greater detail. Okay.

Now it starts getting interesting. It could be argued – I mean I presume that you would know this or have heard of it. I would presume you heard this: generally, Lagrange multipliers are taught as a behavior in an advanced calculus class somewhere where someone says how do you solve that and you're taught the following – then you should do this. And it doesn't make much sense, but you're told that's what has to happen. This would maybe be the first one where it's not obvious, and something maybe you haven't seen. So what are the conditions under which – when would a point x in the non-negative orthant minimize a convex function? That's the question. Well, let's just apply the – we'll end up deriving it, but it's gonna look something like this. It basically says grad f(x)T(z – x) = 0 for all z positive. That's our basic condition. And let's simplify this now. So let's take a good look at this. This has to be positive for any non-negative z here, so let's see what happens. First of all, I can plug in a bunch of things. I can plug in z = 0, and I get the following that grad f(x)Tx = 0. Everybody agree with that? That's from z = 0.

And now I can do the following. I can let z – if an entry of this vector were negative, I'm in big trouble because if an entry were negative, I would take z – if the ith entry of this thing is negative, I take z = t times ei, and then I let t get super big. Then what happens here is this thing here – as t goes to infinity, this thing is an affine function of t with a negative slope, and there's no way it's gonna be bigger than or equal to zero. So we can also conclude that this vector is positive. There's a squiggles there. These are straight. These are squiggled. Okay? But now it's getting interesting because look at this. I have the inner product of two non-negative vectors and it's less than or equal to zero. Now when you take the inner product, you take the sum of the associated products, but obviously if you're doing that with two vectors that are non-negative and you end up with something that's less than or equal to zero, there's only one choice. It has to be zero, No. 1. And No. 2, it has to be complementary. In other words, we actually can say this. So basically, it says if xi is positive, then that correspondent – that component of the gradient is zero and vice versa. If this is positive, that's zero, period. This is called complementarity, and you're gonna see it a lot, or you'll see it a bit. Okay? So that's the condition here as we just worked out this. And these are already kinda – these are not totally obvious. Okay. Next topic is equivalent convex problems, so as I said we're gonna use an informal notion of this. You say that they're equivalent if the solution of one is obtained with modest effort from the solution of another. What we'll do is just give a bunch of common transformations that would preserve equivalence, just examples to give you a rough idea.

Here's one. You could eliminate equality constraints. So I wanna minimize f0(x) subject to fi(x) = 0 and Ax = b. I can eliminate these, and the way I do that is I find a matrix F and a point z so that Ax = b is equivalent to having the form Fz + x0 for some z. Now how do you get F? I mean these are quite easy. F is like the null space. The columns of A would be a basis for the null – not a basis, sorry. They would span. That's all we have to do. They would span the null space of A. And then x0 is any particular solution of Ax = b. For example, it could be the least norms solution for example. So this is linear algebra to go from Ab to F and x0. But the point is this is what people would call a constrained representation of an affine set. And this is what people would call a free parameter representation. So this is parameterized by z which is now free. Okay? If it’s a minimal one, you could say something about the size of these things, but that’s just linear algebra.

So what I’ll do now is I’ll make a new problem which is this. I’ll take x = Fz + x0 and I get this problem here. I don’t needs this constraint because basically A(Fz + x0) = b for all z automatically because AF is zero. That’s how you constructed F, and Ax0 is b. That’s how you constructed x0. So you don’t need this constraint. You drop the constraint, and you get this. Now these are not the same problem. I mean they’re totally different. They have dimension in variables. This one has equality constraints. This has none. But they’re clearly equivalent, and the idea would go something like this. If you wanted to solve this problem, you would solve this problem and then return as x, Fz + x0. That’s what you’d do. Okay? Now the idea that it has to be a modest transformation to do this has something to do with the following fact that when you get A and b, it has to require modest computation to calculate F and x0, and indeed it does. That’s just basic numerical linear algebra.

Student:[Inaudible] F again?

Instructor (Stephen Boyd):Yes? Oh, F – the columns of F are for example a basis for the null space of A. That’s good enough. There’s many ways to construct such an F like that. Okay. And by the way, if this problem is convex, so is this one. And the reason is that the objective is actually a – you pre-compose with an affine function, but if you pre-composed with an affine function, that preserves convexity. Okay? This one is unconstrained, no equality constraints. Now you can do the opposite, and in fact the shock or hopefully within three or four weeks you will – I mean later in the class there’ll be like – I don’t know. When you – somewhere near the ninth week of the class, you’ll think what were some of the shocking things – that a lot of it kinda sorta made sense intuitively, but some of it will be – there’ll be things where you’ll say that wasn’t obvious. Now I’m gonna tell you one. This is just a topic, but we’ll get to it. It’s this. It’s actually often extremely useful to introduce equality constraints, so to “uneliminate.” Everyone has a bias towards elimination because people like to solve problems with smaller numbers of variables. I will cure you of that before this quarter is over. So if you know what you’re doing, not only is it sometimes not a good idea to eliminate constraints, it’s often a good idea to introduce new variables and constraints, which sounds very odd when you first see it, but trust me before the quarter’s over, it’ll be very natural to you. Yeah.

Student: Yeah, why do the equality constraints have to be affine? If we’re just gonna compose it and need to preserve convexity, couldn’t we just have any convex –

Instructor (Stephen Boyd):Well, my answer to that which is – the question was why do the equality constraints have to be affine, so my answer to that is gonna be very unsatisfying to you. It’s this: that’s our definition of a convex problem. By definition, a convex problem – the only equality constraints we have our affine. Sorry. Well, generally speaking, other affine constraints don’t lead – then the constraint set generally is not convex. And then you don’t even meet the broader definition of a convex problem, which is minimizing a convex function over a convex set. So sorry, it's by fiat. Convex problem has affine constraints. I told you you wouldn't like the answer. Yeah. Oh well. Sometimes there's good answers. Sometimes there's unsatisfying ones. That was the latter. Okay. So how do you add – I mean you can actually introduce equality constraints, which by the way some people call unelimination. I kinda like that. So it works like this. You have a problem that looks like this. You could do the following. I'll introduce a new variable y0 here. I'll introduce a new variable yi here. And I'll have this problem. Minimize over x and now also yi, so I've added new variables – f0(y0) subject to fi(yi) = 0, and then these equality constraints.

Now you could reasonably ask – it's entirely reasonable to ask the question why on earth would you add extra ridiculous variables because this doesn't look like progress. The problem started with no equality constraints. After this unelimination step, you have added variables and added equality constraints. This does not look like progress. Later you'll see shockingly this is the first step to lots of progress. I know it's weird, but this will all be clear later. Another trick is to introduce slack variables for linear inequalities. Now that works like this. You're gonna say I have AiTx = bi. That's fine. What I'll do is I'll convert that to an equality constraint. I'll convert that to AiTx + si = bi. I mean it's stupid. This is the same as saying si is the slack. It's bi - AiTx, and then I'll write that the slacks are bigger than or equal to zero. Or to put it in our absolutely canonical form, I would write – si = 0. Okay? Here – to put it this way. Again, you know these are simple. Actually, what's remarkable is that we're gonna look at a bunch of transformations. Most of them are really stupid like these, and it's weird to think – yeah, it's like everything else. You learn about 15 trivial things, and then you put them together in just the right sequence, and all of the sudden something that was not obvious at all will come out. So yeah?

Student: When you minimize that problem, don't you know for a fact that the minimal si will be zero, or s will be zero since [inaudible] push up against the bound?

Instructor (Stephen Boyd):No. Also, that's not true. That's just not true.

Student: So si can be actually finite?

Instructor (Stephen Boyd):Yeah. The slack can be positive. Sure. It depends. It's basically you solve that problem – I mean if you wanna – by the way, each of these needs an argument that they actually are equivalent. So don't just trust me. After you do a few of these yourselves, work these out, you'll see. Also, you'll probably do some of these numerically, and basically if you don't know what you're doing, you'll just come out with the wrong answer, so that'll be another feedback mechanism for you. But let's just check here that they're equivalent. Well, if I solve this problem, then I can define si = bi - AiTx. Now if you solve this problem, it's feasible, so the sis are positive, and those sis are feasible for this, and in fact they're optimal, but I'd have to argue that, which I'm not gonna do. They didn't go backwards. And you can say suppose someone solved this. We're gonna make a – the wrapper that would allow you to solve this problem if you had code to solve this, or if you had the solution for this or whatever. So if I solve this problem, I drop x and I return – the solution of this one gives me x and s, and I simply drop x and forward s, and argue that it's actually – here, it's optimal here. So that would be the argument. I'm actually not gonna do it because these are actually shockingly boring, but should be done once or twice or something or more, but not many more times than that, and you will. I promise.

Student:[Inaudible] x and s – is there another objective? Or what is –

Instructor (Stephen Boyd):No. It just means I have more variables. So it means that when you minimize over x here, it means that x is what you can mess with. Here when you minimize over x and s, it says you have x and s to mess with, so that's what it means. Here's a simple trick. It's called the epigraph trick. It's actually quite useful, and it's widely used in practice. Let's see how that works. I mean it's really dumb. Here's what it is. You wanna minimize f0. You introduce a new scalar variable t, and you introduce this new constraint, which is f0 – this is basically f0(x) = t, and you minimize t. Now this is kinda dumb because if you solve this problem, first of all you'd be an idiot to choose any t other than f0(x) = t. Not an idiot, you'd just be – that t couldn't possibly be optimal is the point because your goal is to minimize t. However, I cannot adjoin this to this problem. It is an equivalent problem to write equals here. That's absolutely equivalent to the original problem. Unfortunately, it's not convex unless f0 is affine. However, with a less than or equal to, it's convex.

It's a really dumb thing. Let me just draw this. Here's a function that you wanna minimize like this. The original problem basically says find the point here that minimizes f0. This one does this ridiculous thing and introduces a new t that's up here like that, and it says that's your feasible set or something like that, and then it says go as far as you can in the direction –t, right? Like that, actually I guess the direction is -1, and you would recover this point here. By the way, this has immediate practical application. This says for example that if you're gonna write a code to solve convex problems, you don't need to – without any loss of generality, you can just minimize a linear objective. That's quite standard. So you look on – if you go to Google and type C source and then some name of some convex problem, you will find it'll just – many times, they won't even say it. It'll just say it solves the following problem, and it's a linear objective. You go hey, I can't use this. My objective is not linear. Well, it's just expected that you know this, so this is just expected. By the way, it's generally not in the documentation which is actually interesting because that's how widely this is supposed to be known. It's also how unhelpful the documentation is. So this is the epigraph trick, and you'll see this used quite frequently. Oh, the other thing you'll hear is this. You'll actually have people say just without loss of generality, the objective can be assumed to be linear because that's a linear objective. It's a linear function of x and t. So that's just completely standard.

You'll also hear people say things like this: they would say linear objective is universal in convex optimization, so whatever that means. It means something like if you can do linear objective, you can do anything. A couple of other equivalences – here's one. You can minimize over some variables, so if you have a problem that says minimize f0(x1,x2) subject to fi(x1) = 0 for example, I could minimize this thing over x2, and that would reduce – that would give you this function, f0 tilde of x1, and that would give me this. By the way, for those of you who have seen dynamic programming for example, that would just be this – a fancy version of this would just give you all of dynamic programming, and basically then there's nothing to say about it except it's just this. Oh, except interesting point – dynamic programming preserves convexity of a problem. You have a problem of a few variables, or in a general case a whole bunch of variables – for example, stages in an optimal control problem or something like that. If you minimize over some of the variables, you have to take account of that in the constraints, the objective, everything, but when you minimize over one family of variables in any convex function, the result is a convex function.

So we can now say something like this. Dynamic programming, if you know what I'm talking about, preserves convexity, so when you optimize over the last stage variable, and end up with an – if the original problem were convex over capital T stages, the T-1 stage problem also convex, and the T-2 and so on. Okay. Just out of curious, actually how many people have heard about dynamic programming and stuff like that? Okay, so not an empty set. Okay. Quasi-convex optimization – well, here f0 is quasi-convex. There are lots of actually practical problems that are gonna be – we'll see lots that are quasi-convex or something like that. And here you have to be very careful because for example this most basic thing which says a locally optimal is globally optimal is completely false. And here's a perfect example. I'm making it exactly flat here. So that's a locally optimal point, no doubt about it. And it is obviously not the global optimum here. And that's a perfectly valid quasi-convex function.

So a lot of things are different now and need not be right. But here's how you solve them with total effectiveness. It works like this. Generally speaking, if you have a quasi-convex function, I mean unless you're just doing a theoretical analysis, if you're actually gonna actually do anything – that is to say construct an algorithm that solves a problem, solve an actual problem, or anything like that other than just talk about it – then what's gonna happen is this: you're gonna actually want – you want a convex representation of the t-sublevel set. So it's not enough to say – so you construct a – now, a convex representation looks like this. It says that you look at this sublevel set, and you must write it as – I've written it as one convex inequality parameterized by t, but you don't have to. It could actually mushroom into a bunch of convex inequalities. It just doesn't matter. You need a convex representation of the sublevel set is what you need here. Here's an example. It's a general fact that if you have a ratio of a convex to a positive concave – actually, non-negative convex over a positive concave function, this function here is actually quasi-convex. It's very easy to show that because – well, I mean there's some horrible details, but let's do this. You wanna know is that a convex function of x.

By the way, if this were a convex function of x and t, that would be the proof that this function is actually convex in x. It's not convex. It's quasi-convex. That means t consider fixed, and you wanna know is the set of x that satisfies this convex. That's the question. Okay? So if you look at this, well you just multiply through, and you get p(x) = tq(x). Now you're in pretty good shape because I write that as p(x) – tq(x) = 0. That's convex. That's concave. t is bigger than or equal to zero without loss of generality because if t is negative, the sublevel set is empty and convex, so I've already dealt with that case. Then this is convex minus non-negative multiple of concave. This whole thing is convex. That's a convex representation. Okay? It depends on t affinely, but that doesn't matter. In fact, the dependence on t is completely irrelevant. Okay? Now the way you solve a quasi-convex problem is this. You just solve it – if you think about what you can do now, if you have this convex representation for any T, it means you can do the following. You can actually ask a question like this. You could say is the optimal value less than or equal to t? And you have to solve a convex feasibility problem to determine it.

So you could say is the optimal value less than three, and the answer could be no. Then you'd start with a value of t where the answer's yes, like t equals 10. Then of course you'd do bisection, so you'd take the average of these. You'd take 13, and then the next query would be 6.5 and so on and so forth. So that would be basic bisection, and that looks like this. So you start with a lower and an upper bound on p*. By the way, you might not have that, in which case sort of a practical version of this would start with t = 1 and query. And if that is infeasible, you would then try t = 2, then four, then eight, then 16 or something like that. And then when you get to a big enough number that you wouldn't be interested even if it were feasible, you would quit. So that would be one way. If it's feasible and you don't have a lower bound, you'd start with t = 1, then you'd try a half. If that's feasible, you'd try a half, a quarter, and when you get down to something very small, you'd quit. Okay. Well, sorry that's all on the assumption that it's a positive function that you're minimizing. So you start with a known upper and lower bound on p*, and you simply query at the midpoint of your known upper and lower bounds, and you solve the convex feasibility problem. So this is just bisection. And every time you do this, your interval – you end up with two points, an l and a u. l is you actually have an x where the – sorry, you don't have an x. For u, you have an x that actually beats – that has at most the value f0(x) = u. And l on the other hand is a value known to be infeasible. There's no x that satisfies f0(x) = l. Anyway, and then at each step that interval of ignorance divides by two, so your ignorance halves at each iteration. So it takes log ten steps, and you reduce your ignorance by 1000, and 20 steps a million roughly. That's the idea. And yes, you'll definitely do this at some point.

Student:[Inaudible] wrote up there has objective zero implicitly?

Instructor (Stephen Boyd):Right.

Student: Okay. And what does the colon equal mean?

Instructor (Stephen Boyd):Where?

Student: In the bisection method?

Instructor (Stephen Boyd):Here?

Student: Yeah.

Instructor (Stephen Boyd):That's an assignment.

Student: Okay.

Instructor (Stephen Boyd):Okay. Now we're gonna start in on – we're gonna look at some problem families. This is actually quite interesting, quite useful. The problem families are interesting mostly because you can actually – all the things we're gonna look at actually are extremely practical. I mean what we've done so far is all very general and follows – actually would follow a traditional course on this material, but it would stay at that level. We start talking – the things we're gonna talk about now are extremely interesting theoretically. In fact, some of them are at the height of fashion in various fields at the moment, not linear programming but other things we'll see. All right? The absolute height of fashion even among people doing strictly theory, and it's very interesting. But actually, what's mostly interesting about it is that all of these things are extremely doable, so you can do this. In fact, you will shortly. So that's sort of the – we're now moving into things that this is more than stuff you talk about. These are things you can actually do. So I just thought I'd point that out. So linear programming – a linear program is almost sort of the simplest optimization problem there is in one sense. All of the functions involved are affine. Everything is affine. The objective is affine. The inequality constraints are affine. By the way, and people call them linear so that's okay. And the constraints are affine, again called linear constraints. It depends on whether when you say affine or not – linear, you allow an arbitrary right hand side. If you allow an arbitrary right hand side, then of course when someone says linear it means affine. Okay.

So that's an LP. By the way, you know this is not a new topic, just to make that clear, not remotely a new topic. So Fourier wrote a paper about this which is a hint that this is not exactly a new topic. In fact, he wrote a paper on how to solve linear programs. It was a method that would be very effective if there were like four variables, but hey that was the beginning of the 19th century. Anyway, so modern history actually has a lot to do with Stanford, but to be fully honest it has a lot to do with Moscow State University and later Stanford, but it has a lot to do Stanford because this was sort of popularized in around 1948 by a bunch of people who were here, like George Dantzig. So the feasible set of course is a polyhedron, so your image of what an LP is should be this. It's a polyhedron, and then the objective is affine, so I mean the level sets of an affine function are hyperplanes, so you're just – here's negative c, and you're told go as far as you can in this direction. And this is a little example makes it totally clear that's the solution. Okay? Now, before you imagine that this picture really fully does the trick of understanding it, I should point a few things out. If you have a linear program with I don't know 30 variables and like 100 inequalities, the feasible set oh it's a polyhedron. It's a polyhedron. Wanna know how many vertices it has? Someone wanna take a guess?

I forgot what numbers I said, so what I'm about to say is a lie. It would become true by increasing those numbers modestly. The answer's just completely astronomical. It's just absolutely astronomical. It goes up very quickly. So you'd write down what's called a small linear program, and the number of vertices it will have on this polyhedron will simply be literally astronomical. It doesn't take much. You can easily – you will solve in this class many linear programs where the number of vertices in this – in the feasible set will exceed probably by a long shot the number of subatomic particles in the universe. You will solve many of them, quickly I might add. So I just point this out because this is the picture you should have. This is what I think of when I think of an LP, but be forewarned that this doesn't really give you a full – when you look at this – you know if you were to go -- if your roommate asks, "What are you doing?" and you said, "It's really – yeah, we learned today how to like go as far as you can in this direction on this set," they'd look at that and they'd go, "When's the drop deadline?" So just watch out. The pictures explain it, but I don't think you get the full idea of it. So let's look at some examples. Here's a famous one. This is just here for historical purposes, so it's the diet problem. So you're supposed to choose a bunch of foods, so you have n possible food types, and you're gonna choose some quantities that you're gonna feed some group of people. I don't know, presumably soldiers. I don't think anyone would eat this willingly, what's gonna happen here, but you're gonna feed some people some amounts of food.

One unit of food costs – you have these costs which are cj, and each one contains aij of some nutrient. That's like whatever 10 grams of the food or whatever our unit is here. And a healthy diet requires at least bi units of some nutrient. So how do you find the cheapest healthy diet? Well, you minimize cTx. Now cTx here is literally the cost because xi is the quantity of each type of food you're gonna put into this mixture. And then when you take the inner product of c, that gives you the actual cost. So that's minimizing the cost subject to Ax = b. A is nothing but this matrix here. It's the nutrient matrix or whatever, and then b is this minimum daily allowed whatever, something like that. Okay? So this is a famous problem. x has to be positive. You can't feed people negative amounts or whatever. So that's the – now, whether you'd wanna eat this diet is another story, and of course it doesn't take much here to realize it's a fairly poor model. It assumes these things sort of add linearly which might not be the case. You might impose – could I impose an upper bound on the amount of a nutrient or something like that here? Of course, I could. That would still be a linear program and so on. But this is just for historical reasons. It's sort of the first one you would hear about. Okay. Here's one: piecewise-linear minimization. So you wanna minimize a piecewise – the maximum of a bunch of affine functions. So you just wanna minimize that over x. Okay? Now, that's an interesting problem because it's not remotely linear, not even close. In fact, when it's linear it's boring because I know how to minimize a single affine function. The answer's minus infinity.

So this is not linear, not remotely. There's nothing linear about that problem. It's also quite useful because actually piecewise-linear functions can be used to approximate actually any convex function. So although this looks kinda silly, it's way more powerful than you think. So just minimize some piecewise-linear function. It's an LP. You just put in epigraph form. So you just write it this way. I'm gonna minimize a new variable t subject to aiTx + bi = t. Done. Now it's linear – now the variables are x and t. These are affine linear inequalities, or to use the term that most people would use, these are linear inequalities in x and t. That's it. By the way, the reason you need to know this – there's some reasons you need to know it because if you went to Google and typed source code

PWL minimization, you'd find some – actually, what you'd find is terrible stuff from total backwater fields where they don't even know it's a linear program. That's what you'd find. And it'd be the worst stuff you've ever – if you found anything.

On the other hand, you won't find anywhere anything that would tell you if you didn't know that piecewise-linear – so I think that's the problem. Everyone knows piecewise-linear minimization is equivalent to an LP. So what you really want to do is get LP solvers of which there is tons. In fact, sadly if you went to Google and typed that, you'd probably go to this PDF file, but the others would be very bad, I assure you. We'll look at another one, and it's the Chebyshev center of a polyhedron. Now we already talked about this a little bit. We talked a little bit about the idea of yield maximization. So yield maximization was this problem. And let me just – this is just to sort of set this up, so you'd know why you would wanna do for example Chebyshev centering or something like that. Yield maximization was that we had some polyhedron of acceptable values of some parameters. We're gonna set a target for manufacturing. So we're gonna set our machines to attempt to manufacture these points. In manufacture however, you will add a random perturbation to this point, and so what you'll get is you'll get a PDF that is centered – has mean value at this point, but otherwise tails off. And then we're interested in maximizing the probability that you're in this set. That would be the yield. Okay?

That's actually generally a hard problem, although convex if the PDF is log concave. But there's actually a simpler method, I mean it's to simply say look, what you really want is you want that point to be well-centered. You want it deep in this set. For example, if the PDF were n zero i, so if the perturbations were Gaussian, uncorrelated, then basically it says that if you look at the PDF, the level sets would be circles. And so basically what you might argue then is you want this thing to be – to find the point here where the largest circle around it fits in this set. It's a heuristic, but it's something that would actually work very well, just something like that for a nice rounded perturbation. So that leads you to this idea of the Chebyshev center of a polyhedron. And what is that? Well, I'd have a bunch of inequalities. I have my polyhedron. And what I wanna do is I wanna find the center of the largest ball that fits inside. So another way to say it is that's the point that is deepest inside the set. I can define the depth of a point in a set as the minimum distance to a complement of the set. It's basically how far you are from the outside. That's the depth of a point. And what I would want to do is I wanna find the maximum depth point. I wanna find the point deepest in this set. Now by the way, you know you're drawing pictures with spheres and balls, the whole thing says find me a Euclidian ball, but that means that – that's not gonna trigger the linear portion of your brain. It shouldn't anyway, right?

So if you talk about linear things, flat is what – the portions that light up when you think about flat things should be flashing. But when I start talking about things like ellipsoids, and balls, and round things, that should just suppress all of the brain activity in the lobe that handles affine things. It turns out that's wrong, so watch this. So I'm just saying that when someone tells you please find the center of the – when you see like ball, you should think round. Round should suppress the linear, and you would not at that point imagine it's gonna turn into an LP. Well, it is and the reason it's quite straightforward is this. aiTx

= bi for all x in this ball centered at xc. That's easy to do. I simply maximize this thing over the ball and see whether that is less than bi. But this is nothing. It's aiTxc. That's the center. Plus, the sup of aiTu when u is constrained to be less than r, and it's equal to this. Okay? Now so this – the inequality that you should stay on one side of the half-space is in fact here, and shockingly it's linear in xc and r. So you can do Chebyshev centering this way. You maximize r subject to aiTxc + r||ai||2 = bi. Now watch out because the norm should also be triggering your nonlinearity detectors. However, the norm is applied to data and it has no effect. The variables here are xc and r, and these are linear inequalities in xc and r. Question?

Student:I don't see why this thing has to be unique, especially with like a regular [inaudible].

Instructor (Stephen Boyd):Why what has to be unique?

Student: The Chebyshev center.

Instructor (Stephen Boyd):Right. It doesn't have to be. You're right. It does not have to be unique, so that's a very good point. However, what this does is if it's not unique, this LP will construct n optimal solution. And by the way, I should mention essentially that's the semantics of an optimization problem.

So if I put an optimization problem down on the table in front of you, all that can be asked of you is to produce a solution. So if someone then comes back later and says, "Hey, wait a minute. I've got two different solutions. That's not cool," it's only not cool if one of them is either infeasible or if one has an objective better than the other, in which case you have to get rid of the person who produced the alleged – who alleged that their non-optimal point was optimal. But in general – By the way, if it really matters to you that it should be unique, then there's actually ways where you can add a secondary objective or something like that to this. But that's a very good point, and in fact for a Chebyshev center, it doesn't have to be. Just take a rectangle, and it's completely clear. Take a rectangle. You can fit a ball in it, but that ball can move. But any of those points technically solves the Chebyshev centering problem. Now for yield maximization, you would further want that ball to be in the center, right? But just to give you – but that's a very good point. Yes?

Student: Kinda like the [inaudible] LP?

Instructor (Stephen Boyd):No, you could – this would be easily generalizable to other norms, very simple. All that happens here is if that's a norm, that's the dual norm. So no – here, I'll just do it right now. There. Done. Now it's general norm. Now it's – you can't call it the Chebyshev. It's the general norm maximum depth point ball in polyhedron problem or something like that anyway. You could probably bet some good acronyms out of that, but we won't do it because we're out of time.

I think the only thing I will say about this is this is a really simple example, but it's actually very cool. Let me tell you why it's cool because these are example – like the piecewise-linear minimization and Chebyshev centering, these are sort of – It should be giving you a hint that a stupid problem family like linear programming that's been around since Fourier, was fully understood by Gauss – actually, I don't know that but it would appear to be a safe statement since everything I think was fully understood by Gauss but that what's cool about it is this kind of very innocent looking boring thing that if you go to the OR department, you'll find out about diets and some other stuff – What's cool is it actually has lots of – and a lot of them are not – they're not obvious things, right? I mean it's not obvious that finding the biggest ball that fits inside a polyhedron is a linear program for the reasons I mentioned before. So that's – we'll quit here. I should remind you that at 11:00 today, we have the section that should've been, would have been yesterday.

[End of Audio]

Duration: 80 minutes

ConvexOptimizationI-Lecture06

Instructor (Stephen Boyd):Great. Okay. Can you turn off all amplification in here? That'd be great. So let me start with a couple of announcements. The first is – let's see. Oh, Homework 3 is posted. I know you'll be very happy about that. Actually, Homework 3 start early. Do not start next Tuesday. Just a hint. Start early. In Homework 3, there's numerical stuff, so although it should fit you. You'll see the X. Although, you're absolutely welcome to go the Python route. Actually, anybody here gonna go the Python route? There was at least one person. Typically, that would correlate with people who wake up at let's say 1:00 p.m., so it's not surprising that they're not here. They are in the class but sleeping right now. So you will use CBX. It should install in about 25 seconds or something like that. But just in case it does not, that's why I recommend you start immediately. And the other thing is like anything else, you just wanna start early and get all the stupid questions out of the way and all that early. So please start Homework 3 early.

The other thing is that the Monday section, Jacob will give, and it's gonna be on using CBX, so you're strongly urged to go to that for this. And of course, we'll post both the video and we'll also post the slides for Monday's section. Let's see. I wanna make one other very general announcement about the homework is we had a lot of questions about what do we mean sort of in the homework, like when it says which of the following functions is convex. And we got people asking things like do I have to have a full proof and all that sort of stuff. So I thought I'd say something about that. The main thing actually is that you should sort of understand it, and work through everything, and try to figure out which things are quasi-convex or quasi-concave or whatever they happen to be. Some of the problems we asked of course, these are not by any means easy, so if you have a headache trying to figure out if the quartile as a function of the probability distribution is whatever it is – if you have a headache from that? That's normal. That's as it should be.

Most of the applications won't involve stuff that tricky, but some actually will, so that's kind of the idea behind that. As for how much detail you have to give to justify this, that's kind of up to you. You're welcome – we don't need to see some long proof or something like that, but actually some justification would be good unless you have developed some kind – well, I guess you all know about the concept of like an idiot savant. These are people who sit in institutions, drool most of the time, and then about once a week look up and spit out a 32-digit prime number. Everyone knows about these people. Apparently not. So I haven't encountered it yet, but if in fact – maybe not such a severe case, but if it turns out you can look at a function, just think about it, make no calculations, just look at it, smile and say quasi-concave, that's fine. Actually, if that's the case – if you find that you have that ability, I'd like to speak to you. Barring that however, the important part is that you should have some kind of a justification. If you happen to have had a class on analysis, or if your background is strong in math, feel free to give us a nice proof. If it's not, don't worry. We just want some kind of a justification. You wanna look at the sublevel sets or something like that. So any question about last time? If not, we'll continue right along. You should be reading of course Chapter 4 by now, and in fact should probably be finished by around today or something like that. So between the homework and the lectures, you should be okay.

If you can go down to the pad, last time we looked at linear fractional program. That's a very famous – and maybe the most famous quasi-convex problem. There are lots of variations on it, and a variation – it's a generalized linear fractional problem is this. You minimize a linear fractional function here. Now always in a linear fractional function, you have to decide which sign the denominator has, and by convention this would be positive. That technically has to be specified. You have to specify the domain. So here's a linear fractional function. You minimize a linear fractional function subject to linear inequalities and linear equality constraints. Now that's a quasi-convex optimization problem, and you can solve it by bisection. That's easy to see because if you wanna know is F0 of X less than T, that's the question. So you're really asking is the optimal value of this problem less than equal or T. Can it even be achieved, the objective value T?

All you do is you take this thing which is positive, and you multiply it out, and it turns out that's equivalent to the linear inequality like this. Now that's a linear inequality, and therefore you'd call it an LP feasibility problem or something like that. To check if the optimal value – if there's an X that satisfies all this and has F0 of X less than T, to solve that – to determine that feasibility problem, you would solve an LP feasibility problem. And then you could bisect on T and solve this. Okay? So that's one way to solve this. It turns out though, some quasi-convex problems can actually be transformed to convex ones. It hardly matters frankly, but in this case it's good to see how this works. And let me point something out. If you were to take this approach, you'd have to solve an LP for each bisection step. Now the number of bisection steps you'd probably have to solve would be somewhere between 10 and 20. 20 would give you two to the minus 20 in bisection, and you would reduce your ignorance – that is the difference between a known feasible point and a known bound – you'd reduce that by a factor of a million. So you'd have to solve by 20 LPs.

By the way, for most things it wouldn't make any difference at all. And it's even more than that. Later in the class, you'll understand that you can actually do something called warm start, which means you actually solve one LP, which is very close to the LP you just solved, and it can be done very fast, so the effort is not even a factor of 20. It's probably a factor of four. Nevertheless, it's interesting I think theoretically, and it's actually interesting in practice that this quasi-convex problem can be turned into actually just an LP. Yeah?

Student: That second line, should that C transpose X plus be on our right side, should that be a [inaudible]?

Instructor (Stephen Boyd):Should it be C transpose – in fact, what on earth should it be? Should it be E transpose like that? Are you happy now? Good. You're happy because what I had was wrong, and this is right. That's good. Thank you. It was supposed to be the denominator. It just didn't work out that way. So let's see how to do this. It's very interesting, actually. It turns out this is nothing but the perspective transformation. Or some people call this homogenization, so let's see how that works. What you do is you write down this problem. And let's actually see what happens here. What you do is you write down something – well, here. It's equivalent to this LP, and let me go through the argument, or at least I'll give you some of the argument. The full argument you can find in the book.

Here what you do is you introduce a new – I start with the original problem. Let me just start with the original problem up here, and I'm gonna introduce a new variable called Z scalar. I hope this'll work. So I wanna minimize C transpose X plus D divided by – this time I'm gonna get it right – this. I wanna minimize that. Note the period. Everything's cool here. Subject to – that's st, but it means subject to of course – H and AX equals B. Now you know when you do things like you have a quadratic function and you add a new variable that's equal to one to convert it to a quadratic form, I guess that general principle is called homogenization. You make the problem homogeneous by introducing a new variable. So the way we're gonna do that is this. I'm gonna introduce a new variable called Z, and I'm gonna say it's one. So I haven't hurt anybody, and I can most certainly do this. I guess the Z should've gone in front of the H, but you know it's okay. In loose parsing, you can put a scalar after a vector. Everybody cool on this? I mean how could you not be? I just multiplied a bunch of things by one.

However, if you stare at this problem long enough, without the Z equals one you realize the following: it's completely homogeneous. In other words, scaling X and Z up here by a positive number has no effect whatsoever on the inequalities, none, has no effect on the objective. That means in fact I can normalize – my current normalization just equals one, but I can normalize it any way I want and simply divide out later. It will make no difference. So I'm gonna renormalize this problem. And I'm gonna renormalize it – now I am cheating. I'm looking down here. I'm gonna normalize it so that this – I'm gonna comment this out, and I'm gonna add this constraint in. That's just a different normalization of the same thing. It's just fine. Now – what's that? Z has to positive. That's correct. Sorry. Thank you. Actually, technically Z has to be strictly positive, so in fact I'll even write it that way like that. Now if this is one, the denominator goes away. That goes away. And I am left with an LP here that I have to solve. There's a question.

Student: Sorry. What's Y?

Instructor (Stephen Boyd):What is Y? It's nothing.

Student:[Inaudible] even shows X?

Instructor (Stephen Boyd):No. It's just nothing. It's supposed to be X. It's just one of those days, I think. I have a feeling where yeah, fine, good – keep me on my toes here, not apparently that it's gonna do any good today, but we'll see what happens. I should've had that second espresso. I think that was the problem. I didn't. I meant to, but I didn't. All right. This is now an LP here. And in fact, if you replace this with this, you get all sorts of information out of this. I won't go into the details, but for example if you were to solve this and Z comes out positive, then by simply renormalizing here, you get this solution. That's the first part. If Z turns out to be zero, that's like this one case where this thing is unbounded below or something like that, but that's the idea. By the way, this is not a simple trick, but it's actually one that comes up a couple of times, and it's pretty cool that you can do this. By the way, for most quasi-convex problems this trick won't work, but for some it does.

Now we get to the idea of a generalized linear fractional problem, and that's this. You wanna minimize the maximum of a bunch of linear fractional functions. And that's on the polyhedron where all the linear – well, it's at the intersection of the domains of the linear fractional functions. That's quasi-convex. There's no way to convert that to an LP in general, so that you're gonna solve by bisection. And here's an example that's actually quite interesting. It's the von Neumann growth model, and it works like this. We're gonna optimize over X and X plus the following. So X is gonna be a set of activity levels. I guess we have to have X positive here too, or something like that. X is a bunch of activity levels at one period, and X plus are the activity levels in an economy at the next period, so that's what these are. Then we have matrices AX and AX produces – this gives you the amount of – AX tells you the amounts actually produced by an activity level X, so this could be for example in our M, if the N activity levels produce say M different classes of goods. So if you have an activity level X, AX is a vector telling you how much goods – the vector of goods produced by that activity level here. That's AX. Now when you operate the economy at the next step, that's X plus – are the activity levels. BX plus – that's a vector telling you how much is consumed. And then you simply say that the amount consumed at the next stage here, it uses some – it has to use the goods produced in the first stage, so you'd have an inequality that looks like that. Okay?

These describe – it's a polyhedron, of course. Right? The set of activity levels current and next that satisfy these inequalities is a polyhedron. And now what we want to do is we wanna maximize over these the minimum activity growth level. That's XI plus over XI, so that's what we want. I guess we don't need this because when you write this thing, X positive is implicit here. To maximize this is the same as minimizing the negative of this, which is minimizing the max of XI over XI plus, each of those, so that's the same thing. And that's exactly of this form. That's a quasi-convex optimization problem, and you can solve it directly, so that's one. What you wanna do is you wanna set the activities in order to maximize the growth rate of the slowest growing sector, so that's the problem here. And there's lots of variations on this.

So that ends up sort of problems that are linear and related to linear. Now we move on to quadratic. By the way, I can tell you what the year is on some of these. The linear programs as I mentioned earlier go back to Fourier at least, but the year maybe generalized linear fractional stuff is maybe the '50s or something like that, already widely used in the '50s. Quadratic programming is the same. This would be mid-'50s. So in a quadratic program, you have a convex quadratic objective. That's here. So P is positive semi-definite, and you minimize this over a polyhedron, so a set of linear inequalities and a set of linear equalities. By the way, if you just said P equals zero, you recover an LP, so this is a strict extension of LP. And the picture now is this. Here's your polyhedron which is your feasible set shaded, and your objective function now instead of being affine or linear is in fact quadratic and convex quadratic. So the sublevel sets are ellipsoids, and so this is showing you the sublevel sets of a quadratic function. Of course, the gradient points down the slope, and is the normal to the tangent hyperplane at the level surface. So for this particular problem instance, that's the solution there. One thing you see immediately that you didn't have for an LP – it's a basic fact, although not of particular importance for us, that in a linear program you can always take a solution that's at a vertex. This is very basic. It's kind of obvious geometrically. It's also completely obvious analytically, but it doesn't seem actually to have too much use for us and in most cases, although it's made a big deal of I think mostly for historical reasons.

But you can see clearly here that the solution does not have to be at a vertex. It can be inside a face. Again, it doesn't make any difference to us or whatever, but that's – so this is a quadratic program. We're about mid-'50s here in where we are historically. So let's look at some examples. The most basic one would be just least squares. There's no constraints. That's basic linear algebra. The solution is just given by – or sorry, a solution I should say in the general case is given by this pseudo-inverse or Moore-Penrose inverse. That's just A dagger B. But immediately for example, you can take a least squares problem, and a very common thing – I mean this is really dumb, but you just add some linear inequalities. In fact, [inaudible] balance. So that's a quadratic program. By the way, what you're going to see later in the class is that for example a least squares problem with bounds on the Xs, that can be solved basically as fast and as reliably as just a least squares problem with a very small modest factor, totally reliable.

Let me actually say something sort of culturally about how the course goes. There's a lot of people that know about least squares, okay? A lot. I guess not a huge number, but you know there's a lot of fields where least squares is widely – a lot of people know about least squares in signal processing, statistics, communications, machine – it goes on and on. A lot of people know about that, okay? You just throw in something as dumb as this, ranges on the Xs, and that number of people who recognize that as basically a problem that is completely solved is – just technology. We can just solve that, period – drops by at least a factor of ten. What you will find instead – do people have things like this? Sure, they do. But of course, what they do is they do heuristics. They either invent their own algorithm that's impossibly long, might actually work, but it's fantastically complicated, or they make some iterative thing with regularization. You wouldn't believe what happens when people encounter things like this. And people make a big deal out of it. There's no reason for it. So I think the zeroth order thing to do is to look at a problem like that and say QP, therefore we're done. It's just done. There's almost nothing interesting to say about it. You'll write code that solves this. It'll be 30 lines. It'll be totally reliable. So it's not a big deal, but at the same time it has huge practical uses.

So if you wanna do parameter estimation but you have bounds on the Xs – here's a stupid one. Suppose you're estimating powers or intensities. Well, these are generally speaking non-negative, so that's just like non-negative least squares. So for us, that's just a QP. I'm telling you now, that's already – you've moved into a much smaller group than the people who know about least squares. For example, a whole huge thing in statistics – this is just amazing actually to me, but whatever. So that's the constraint that the Xs be ordered. So of course that's a set of N minus one linear inequalities. I think you had it on some homework, like Homework 1 or something. I don't know. Write us a set of linear inequalities. So believe it or not, there's a whole subfield called isotonic regression or something. There's books written on this, literally on minimizing – on doing least squares with that. Why is there a book on it? Because there's no analytical solution like there is for just ordinary least squares. It comes up enough anyway. For us, monotonic regression? That's below the level I would even assign as a homework problem. No, I would go that low. But it would be one of a – when we're counting up the amount you have to do in a week. That would count for like none, very little anyway, and whole books on it.

So I'm just mentioning these things are simple, but actually so far this has not diffused out to a lot of application areas because people haven't gotten the word yet, which is very strange to me, that you can do this and it is completely straightforward. Okay. We're back on track here. Here's an interesting one is linear program with random cost. So that's another – let's do a specific example. If you wanna make a specific example, let's do diet – the infamous diet problem. So you are – X represents the amounts of a bunch of foodstuffs you're gonna buy, something like that. And there are constraints on it. There are things that have to balance out exactly, and there's inequalities. Of course, this can include lower bounds and upper bounds on for example nutrient levels and things like that. So our constraints become a polyhedron in X here. Now what we're going to – the problem now is we're gonna formulate this diet now, but we're gonna implement it next month, and we don't know the prices of the foodstuffs, so it's not as simple as saying just minimize C transpose X which would give you the cheapest diet that would meet the requirements because you don't know C. You should know something about C. C for example has a mean, C bar, but it also has a covariance sigma, something like that. This doesn't matter here. There are many things you could do. In practice, I'll tell you what would probably be done would be that. You would simply ignore the variation and just work with the expected values.

Now it's not hard to imagine that that could be a seriously bad choice because it's very easy to do. All you do is take one foodstuff, or one component, or whatever and you make it slightly cheaper than another one that has similar nutrients, but you make it have much more price volatility than the other one. Everybody see what I'm saying? The LP will – I mean if you ignore the risk – this is price risk – if you ignore the price risk, the LP of course will use the cheap stuff based on this expected value. So that's an extreme example, but that's the kind of thing you could do. So what you really wanna do is trade off mean price and price risk. This would be done – you'll see we'll do more of this later – by simply minimizing a weighted sum of the expected cost and the variance here. Gamma is a risk aversion parameter, and there's lots of ways to figure out what it is. For that matter, you would probably solve this problem for a whole family of gammas and get a tradeoff of expected price versus the variance in the price. And then you select based on your risk aversion where you should operate on that curve. So that's the point. Okay. But if you work this, take a look at it, it's nothing but a – this is a QP. There's one condition. Gamma has to be positive here. Otherwise, this is no longer convex. So what's interesting about that is that you can do risk averse design by convex QP, but you can't do risk seeking. If this were negative, it would be risk seeking. You'd actually prefer if there's two diets that have about the same mean cost, meet all the constraints, in risk seeking you'd actually prefer the one that has higher variance in cost. Fortunately, that's not something you wanna do. So it turns out once again what you wanna do lines up with what you can do.

I should make one other comment here, and that's this that in general – by the way, not entirely always – if P is not positive semi-definite, so if it's got just literally one negative eigenvalue or something like that, this problem goes from being extremely straightforward to NP-hard. So if you know what I'm talking about, that's fine. If you don't, I will just say this. It goes from a problem that is in theories – actually multiple theories would assert that that problem is now very hard to solve, and also in practice. You'd go from a problem where you can solve the exact solution if X can be huge like 10,000 by – that's just no problem. The 30-line code you'll write to solve this for example will easily solve that. If this becomes indefinite, oh, you can solve these problems, but for example with X in R20, you'd have to have a lot of luck and a big old cluster, and you might get the answer later that day. You're just in a completely different regime. If X is in R100 when P is indefinite, this problem essentially becomes impossible. So convexity makes all the difference with quadratic programs. Okay.

Now another generalization – I mean it's kinda silly but – is a QCQP, that's a quadratically constrained quadratic program. So here the constraints can also be convex quadratics. Now here, if these are zero, you recover a QP. And of course, if all the Ps are zero, you recover a general LP. Here, these things – if P is positive definite, of course this represents an ellipsoid, so here the feasible set is an intersection of – well, ellipsoids and degenerate ellipsoids. Now, a degenerate ellipsoid is one that can have a flat face and be infinite. So a half space is actually degenerate ellipsoid, for example. But for example, if P is positive semi-definite but only has Rank 3, this is sort of a degenerate ellipsoid. It means that in N minus three dimensions, it's sort of infinite. In three dimensions, it's got curvature. For example, it would be a cylinder in R2 for example – R3, sorry. A cylinder in R3 would be a degenerate ellipsoid. That would be degenerated by a Rank 2 – a three by three matrix P that's Rank 2, and this inequality would then be not a cylinder, but it would be an ellipsoid like this, and then it would have an infinite extent in another direction. Okay? So that's it. It's a convex problem and can do this.

Now we get to the first thing. Again, this is maybe – we're still in the mid-'50s. Now we're jumping ahead. Now we get to – well, it depends how you wanna count the history, but you're into the '90s now, so this is new roughly. Although, actually there's papers written in the '50s that refer to this. They knew how to do it when there was one inequality like this, but not in general. So let's take a look at this. A second order cone program – this is SOCP. There was a confusing while by the way in the '90s when these were referred to as – some people were trying to call these QPs and it never stuck. But there was a brief period with a small group of where they tried to get this to be called QP, but it just didn't – fail. And it seems like SOCP has won out as the term. So here it is. You minimize the linear function subject to – and the constraints are quite interesting. It's a norm of an affine thing is less than or equal to an affine thing. Notice – very important here, it is not norm squared. If this was norm squared, this problem would be a QCQP. It is not norm squared. This sort of the square root of a convex quadratic. That's what this thing is. Okay?

And it's called second-order cone programming because each constraint is nothing but this. You have an affine mapping that maps X into AIX plus BI, that's a vector, comma, then a scalar – CI transpose X plus DI – and that that should be in the unit second-order cone, the Lorentz-cone in RN plus one. Okay? So that's each of these constraints. Now this is more general than everything we've seen so far. It includes – well, not the linear fraction. It includes linear programming. That's obvious because you just make that zero here. It includes linear programming. You can rewrite quadratic programming and you can rewrite QCQP in this form. But it's more. There are problems you can write as an SOCP you cannot write as any of these, so this is a real extension. There's also something very interesting to point out here. If you write the – if you rewrite this in sort of our standard form, that's AIX plus BI minus – I guess you'd write it this way, right? Minus CI transpose – minus DI is less than zero. So this would be FI of X. Okay? That would be our standard form to write this. Here's a very interesting fact about FI of X. It's not differential. This is the first problem we've seen that has that feature. It looks differential. I mean – well, it actually – this function – of course, that's affine. That's differentiable everywhere. By the way, when is the norm differentiable? The two norm? Where is differentiable? Where is it not differentiable?

Student:[Inaudible] is the origin.

Instructor (Stephen Boyd):It's not differentiable at the origin. How about everywhere else? It's analytic everywhere else. It's got derivatives of all orders, right? And in fact, just think because of the graph. The graph is like this ice cream cone. It's perfectly smooth, no problem except for that tip.

So by the way, you might say then, "Oh, come on. That one point makes that much difference?" This is like what are we in the math department? Only a mathematician would worry about that – these are the kind of things you would think but not say. You would say what kind of a human being would worry about this one point where it's non-differentiable out of all RN. What do you think of that? Let's not worry about it. Well, you know obviously that argument is totally wrong. It turns out what is interesting about second-order cone programs is that of course the solution is very often at the point. Obviously, if you have pointed things and you optimize things like linear functions, of course you end up at the point. So yes, these functions are non-differentiable at only one point. Well, okay at the whenever AI transpose X plus BI is zero, it's non-differentiable. However, it turns out that's where the solutions lie. That's what makes this useful in applications and so on. So it's not just something – well, people in math are correctly interested in the non-differentiability. So this is a non-differentiable problem.

By the way, it would be completely unknown how to solve this efficiently let's say in 1988 or something like that. There were some people who knew about it by 1988 in Moscow. That was about it. So now it's completely routine, so it's as if it's always been here, so people do this all the time. By the way, I think it's in Excel now, so to give you a rough idea of what's happened. Well, a lot of these things actually are. Okay. Let's look at an example of second-order cone programming. So very interesting is this – let's start with a linear program. Let's minimize C transpose X subject to AI transpose X less than BI, and what we're gonna do is we're gonna add the idea that there's uncertainty in C, AI, and BI. And in fact, you know what? We'll worry about uncertainty in the A and B. By the way, earlier in the diet problem, we're worrying about cost variation. I'll just make C fixed, and I wanna worry about variations in AI. By the way, if you wanna go back to the diet problem and figure out what that would mean, what would it mean that there would be variations in AI? I mean in terms of the diet problem. What's the implication? What would it mean? What does it mean.

Student: Maybe there's variation in how much nutrition you absorb from certain food.

Instructor (Stephen Boyd):Exactly. That's exactly right. So the point is when you get some foodstuff and you analyze it on Monday, and Tuesday, and Wednesday, it's close but it's not the same. And therefore, if you compose a diet actually, and you ask for so many calories or whatever, and you form it this way and you just use the mean, you might not actually on a sample by sample basis actually have the number of calories you want, in fact the number of calories in distribution. So that's exactly what it is, so it might be nutrient variation or something in the foodstuffs. Okay. Actually, pretty much any LP that's ever solved, here's the way it works. The coefficients in AI, the actual numbers in it – by the way, often they are zero. And by the way, when they're zero, they probably really are zero. It basically means if the third entry of A1 is zero, it means that X3, the third variable, doesn't enter into the first constraint. And the ones by the way often also have the flavor of being actually one because you're sort of adding something. And typically the minus ones, too. Actually, when you see a minus one, it typically really means minus one.

Any other number you see is probably – probably traces back to – if you trace the provenance of that number, it will go back to something which is not known. It will go back to some analyst or some estimate of a price. It'll trace back to some geometry and some Young's modulus or whatever. It'll trace back to all sorts of crazy things. But for sure, any number like that has some variation. You might know it within 1 percent. You might even know it within 0.1 percent. It depends on the field and application, but it's not at all uncommon that you would know it to like 10 percent for example. So that's just to point out that this is quite realistic to have this – to model variation in data in a problem. Now you could do this lots of ways, and later in the class we're gonna look at this in much more detail, but this is just for now just an example of SOCP. Two common ways are this, and it's strange people argue about which is better, and the answer of course is it depends on the application. So in a deterministic model, here's what you would do is you would say something like this. You would say that each of the AIs lies in an ellipsoid. That would be something like an uncertainty ellipsoid. And you would insist that the constraint hold for any AI in the ellipsoid. And so the other names for that would be like a worst case or model or something like that.

By the way, the people who would do this would go and give you long philosophical arguments. They'd say the greatest advantage of their approach is that they don't assume a probability distribution because there aren't any and blah blah blah. They go on and on with this. Or it's a worst case, and it doesn't matter. Detractors would say that's way too conservative because you're assuming the worst thing that when you buy all these foodstuffs and put them together and blah blah blah – anyway, it all makes no sense. And then the other model of course is stochastic. So here you'd say AI is a random variable, and this is called a chance constraint. You'd require that the constraint be satisfied with some reliability. So [inaudible] might be 0.9, more commonly 0.95, 0.99. You might even see 0.999. And by the way, the embarrassing thing is that these two are not that far apart because for example if that's 0.999 and you have a three sigma ellipsoid and you put that over here, you're very close to doing the same things, but it doesn't keep these people from – they still fight with each other. So that's the – and it's very strange because it doesn't make any sense at all. It depends on the application, what's the cost of violating the constraint, and so on and so forth. It all depends.

So [inaudible], both of these turn out to be SOCPs, second-order cone programs, and the way you do that is this. Let's parameterize the ellipsoids as a center plus the image of the unit ball under a matrix PI. So this is one parameterization of an ellipsoid. And then the robust LP looks like this. It says minimize C transpose X subject to AI transpose X is less than BI. That's for all A in this ellipsoid. By the way, some people make a huge big deal out of this. They call this constraint here a semi-infinite constraint. Got it? Why is it semi-infinite? Because you see this represents a single linear inequality for each element of an ellipsoid, and there's an infinite number points in an ellipsoid. Did you know that? So that's why this is a semi-infinite thing. Big deal. Right? Although it doesn't keep people from writing whole books on it. I'll go on. But this is easy to – when is it true that AI transpose X is less than BI. Let's fix X. You fix B. A varies over this ellipsoid. When would this hole for every element in the ellipsoid – you have to check whether AI bar plus PIU transpose X – whether that's less than BI for all U of norm less than one. Okay? But that's easy to do because this is nothing but AI bar transpose X plus – and then I'll make it this way. I'll write it as U transpose times PI transpose X like that. Now if I wanna maximize this over all U with norm less than one, that's a constant. This is an inner product of U with a vector. And so obviously by the Cauchy-Schwarz inequality that this thing, the largest that number could be is the norm of PI transpose X. And by the way, the worst U would be PI transpose X divided by norm PI transpose X. That's the worst U. Okay?

So I plug in the worst value and I get this, and I insist that that should be less than BI. And that is right here. Now if you stare at this closely, you'll realize that's an SOCP because you have just what you – you have a norm, two norm, of an affine function, and a linear, and a constant. It's SOCP. By the way, let me tell you a little bit about what the implications of this are. People can solve SOCPs now basically as fast as LPs, so almost as fast. Certainly for all reasonable – for huge numbers of situations, it's the same speed, same reliability, everything. That means people can solve robust LPs basically at no additional cost – little additional cost over an LP. That has lots of implications. By the way, it is – that fact or observation is propagating out, so it has now hit finance. So that's why SOCP solvers are in Excel now because whenever you solve an – basically, whenever you solve an LP, then you ask the person how well do you know the data. If they don't say perfectly – and if they do, they're probably a liar – but if they don't say perfectly, then you should say then you should be solving SOCP. And actually, a lot more people these days will say that's exactly what we're doing.

So this actually has important – lots of practical consequences. It's not fully diffused, but it's getting there. Let's look at the stochastic approach. So in the stochastic approach, we'll assume that the constraint vector is Gaussian with mean AI bar and covariance sigma I. Now when you form AI transpose X, that's just a scalar Gaussian random variable. It depends on X of course, but we're fixing X. It's got mean AI bar transpose X and variance sigma X transpose sigma I sigma. And so the probability that you meet your constraint is equal to the CDF. That's a normalized random variable of course. Right? That's BI minus AI – this is the probability. That's the normalized random variable or something like that. This one. And then you put BI here, and this gives you the probability that you meet this constraint. Okay? By the way, these are called chance constraints, and problems that generally involve constraints that involve probabilities of something are – that's a whole field called stochastic programming. This one actually is [inaudible] enough to actually in my opinion deserve a field name. That's not true of some of the others, but this is actually complicated stuff, and I'm okay that there's whole books written on this.

But in this case, in this simple case we can write this out analytically as this, and it's an SOCP. Actually, it's interesting. It's an SOCP provided the inverse CDF of a Gaussian evaluated at eta is positive. That happens for eta bigger than 50 percent. So actually, it's extremely interesting. We can do chance-constrained linear programming. In other words, I can have a linear program with Gaussian random constraint vectors, and I can impose the constraint that each constraint is satisfied with a certain probability. The important part is that probability has to be more than 50 percent. By the way, which corresponds exactly to risk aversion versus risk seeking. So the one we're really interested in probably is eta equals 0.95, 0.99. These are the ones we're really interested in. We're probably not interested in 10 percent. That's risk seeking. Anyway, that doesn't matter because we can't solve a problem for 10 percent risk aversion anyway. Okay. Our next topic is geometric programming. By the way, for all of these there's no reason – you should have read the chapter, so this should not be the first time you're seeing these things. You should not be following everything I'm saying because I'm going fast, but I assure you all of these problems you will encounter multiple times during the rest of the quarter, so you should just be kinda getting a rough idea here. You'll see all of these again. So don't be concerned if – I'm going fast. That's all I'm saying.

Geometric programming – this is an interesting special class of – we'll see they transform to convex problems. This also by the way has a long history. It goes back into '70s. Actually, this goes to the '60s, including the first book written on it. It's super cool. It's this book where they – it's a whole book on geometric programming. I'll say what it is in a minute, but in the last paragraph in the book it actually says – literally like in the last paragraph it says, "You know, it's entirely possible that some day these problems could be solved using a computer." No, I mean it. It says that. The whole book is about these cases where you would solve it by hand by various ways, but it's very cool. By the way, they had it exactly right. So this has a long history. Actually, probably elements of it were known in statistics in the '60s. And it comes up in a lot of other fields. It's a weird thing. When you first see it, you'll think, "I have never ever heard of such a thing," but once you start looking for these, they're like everywhere, or in certain fields they're everywhere.

All right. Here's what it is. So unfortunately, there's some deeply unfortunate nomenclature here, but it came from the '60s and nothing we can do about it now. Just for the record, right – a monomial – people in math have been using the word monomial for like I don't know 400 years or something like that? And it always means the same thing. It's a product of a bunch of – it's a single term polynomial. It has always meant the same thing. There's absolutely no idea as to what monomial would mean, but they decided they would take this term which everyone agrees on has a meaning and extend it. So in the context of geometric programming or GP – you have to be very careful with GP, though. If you type GP into Google, you're going to get several things. You're going to get geometric programming, but you're also going to get something called genetic programming, very different. I'd better not say anything about it. We'll just move on here. So in the context of GP as in geometric programming, you have something called a monomial function. It's a local definition, so don't ever say this outside in public or something like that. Don't ever say that like X to the one half is a monomial because if there's anyone – I mean unless you're only around people known to be talking at that moment about geometric programming because you'll sound like an idiot. It'd be like changing the word polynomial or something like that, and saying, "Well, that's what you call a polynomial, but I call a polynomial this."

So here's what's a monomial. It's a positive coefficient times a product of variables, each one to a power, and the power can be anything. It can be an integer, it can be irrational, and it can be negative. So that's a monomial. This does come up in a bunch – I mean obviously it's clear this comes up in a lot of engineering. People call this a scaling law or something like this. It depends on the field you're in. It's a scaling law or something. Now again, don't look at me. This was not my idea just for the record. They came up with this thing called a posynomial, which first of all sounds stupid, No. 1, and it also is stupid because it's supposed to combine positive and polynomial. That's okay fine, but the point is these aren't even polynomials because the exponents here can be like minus 0.1, and they can be 1.1, so anyway I guess languages are living, and that's a stupid name and it stuck, so posynomial. There it is. It's a sum of these things. So you know an example would be something like this, okay? There. That's a monomial. Okay? That's a monomial, and here's a posynomial. X1 X2 plus square root X2. There you go. And there's a posynomial.

Here's a GP, a geometric program. You minimize a posynomial subject to some posynomial inequalities less than one, and a bunch of monomials are equal to one. Okay? Now you might ask why the one as zero had been our standard before. There's a real good reason. The theory of a GP with right-hand side zero is a very short theory because monomials and posynomials are always positive. So the branch of GP where the right-hand side was a zero here didn't go very far because all such problems are infeasible. So that's a GP. Now by the way, this is not remotely a convex problem. For example, you could say minimize square root X and finish with some stuff down here. That would be a GP. Obviously, it's not convex. Square root X is concave. Okay? So these are not convex problems. However, these can be changed by a change of variables. These can be converted to convex. And this is it. It's actually quite interesting, the conversion, and it's not at all unexpected. So if the variables are positive, you should have at least a slight urge to take the log. That would be normal. So if the variables are positive – we'll just take these variables YI so that XI is E to the YI. And then let's see what happens. We're also gonna do the same thing – we're gonna minimize. Well, you can put log in front of anything you minimize because log is monotone, and I could put log FI less than or equal to zero, so now our friend zero is back on the right hand side. Okay?

But let's see what happens. What is [inaudible] zero? And I'm gonna write this in kind of a – I'm overloading some notation. That's the vector whose entries are E to the YI. So the question is what is this thing. Well, let's take a monomial, and let's take the log of this monomial. Well, you get log C. That's B. Then you get plus, and then you get the exponents. You get A1 log X1, but log X1 is Y1, so when you take – when you change variables by this logarithmic transform, and you take the log of a monomial, you get an affine function. That's good. Now take a posynomial. So here I take the log of a sum of these things, but that's the same as log sum X of an affine function. Okay? Because each of the Xs I replace with E to the YI like this. Then I take a sum of those things, and I take the log. That's log sum X. That's this thing. And if you look at this, this function is convex in Y because it's log sum X of an affine function of Y, so it is definitely convex. That's not remotely obvious by the way here, not even remotely obvious. I mean it is once you've seen it and all that, but – in fact, articles on geometric programming that didn't understand that it converts to convex programming by a change of variables persisted until like the early '80s. So this is not so obvious. So what that means is this GP over here which is not remotely a convex problem – I mean also look at the equality constraints. The equality constraints would be things like X1 to the 0.3, X2 to the minus 0.7, X3 to the 1.2 is equal to one. Well, that has no place in a convex problem. The only equality constraints you can have in a convex problem are affine – are linear. But it transforms to a convex problems like that, so it just transforms right to a convex problem, and is actually it turns out – by the way GP comes up in lots of fields. It comes up in statistics and machine learning all the time. It has to do with exponential families and maximum likelihood of estimation. We'll also see that it's intimately connected to entropy type problems. We'll get to that. It comes up in information theory, but it also comes up in circuit design and seems to be completely ubiquitous. And it comes up in power control and wireless. I'm just mentioning this, and we'll see a lot of these later.

Student: How did you get the monomial F to transform into the log is equal to [inaudible] in between?

Instructor (Stephen Boyd):Can you go over to the pad here, and I'll go over that real quickly down here? So the question was how did I get – if you go down to the pad here, I'll just work this out. Okay. I'll just describe it. You take the log of a monomial, and the first term – oh, you can see it. I see, but maybe it's not on – okay. That's fine. If you can see it, that's fine. But for those of you who are sleeping right now, this is what happens when you sleep through the class. There we go. All right. What we're gonna do is we're simply gonna take the log of this. That's log C plus A1 log X1 plus dot dot dot, plus A2 log XN, but log X1 is Y1, so I get exactly that. That's how that transforms. And what I was saying was that it comes up in a lot of applications. Once you're sensitized to GP, you'll start seeing it all over the place. In fact, kind of in any problem where the variables are positive, they represent powers, intensities, or something like that, densities, you can start thinking how GP might come into play, and we'll definitely see lots of examples of these, of GPs. Actually, it's quite interesting because they're quite natural. For example, in wireless – and we'll see examples of this, but for example in wireless power allocation or something like that, the Xs are gonna be the powers of transmission, or they could be like the powers allocated to channels in some big multichannel system or something like that, or different frequency bands. And what this transformation says – I mean if someone's doing optimization, they say you know what it's better to work with the log of the powers, but that's what engineers have been doing for like decades and decades because these are called decibels. So you work with – so it says you should work with decibels. No one would be surprised by that.

And then the constraints would be things like if signal to interference ratios should exceed like some factor. When you take the log of the constraint, you're basically saying that you should write your constraint this way, that the signal to interference ratio should exceed eight decibels. Again, that's how they would do it. People don't the SIR has to be bigger than like 2.8. They say it's gotta be bigger than you know 8 dB or something like that. So it's actually interesting that exactly the way people who would actually work in these fields would use these things turns out to be – corresponds precisely to the convexifying transformation of variables. We'll see lots of examples of this. Let's look at a mechanical example, and it's design of a cantilever beam. So here you have a bunch of segments. This one just has four, but you know obviously you'd have lots more. And what we're gonna do is this. It's gonna be – so it's gonna stick straight out, and a force will be applied to the tip, and two things will happen. Of course, this thing will deflect. We're not gonna do the huge horrible deflection. We're gonna do the small deflection so that we can use linearized approximations. So it'll deflect substantially, but we'll assume we'll use linear models and things like that. It will deflect, and there'll actually be a strain – a stress – well, both on each segment. Okay?

And what we wanna do here is obviously if I design – the thicker I make the cross-sections, obviously the stiffer the beam. Yeah. Stiffer meaning if I apply a force here it deflects less and the stress is less. So if I make it a big thick thing all the way out – and you can easily – and you probably have a pretty good idea intuitively of what you want here. You certainly wouldn't want a beam that looks like this, that tapers out like that because now you're putting a whole bunch of weight over here. Actually, you're just making it much worse. You'd probably want one that tapers like that. That's just completely intuitive. So the question is gonna be to get the optimal tapering of a beam. That's a cross-section. We're gonna design the shape of a beam, cantilever beam. Okay.

So we'll minimize the total weight subject to – and what we'll do is we'll – and we're gonna actually – these are not – these have a width and height, so there's a height. I know very little about mechanics except I do know that sort of the strength of – well, from carpentry that the strength of thing goes up like the cube of the height, for example. Beyond that, I'm sure there are people here who know a lot more about this than I do. So I've got lower bounds on the width and the height of the segments. We'll limit the aspect ratio so you don't get like an infinitely thin thing like this. We'll upper bound the stress, and we'll upper bound the vertical deflection at the end of the beam, which is actually where the maximum deflection occurs. So that's the same as – Now, this is actually quite a complicated problem. It's highly nonlinear and so on and so forth. So but let's say it's actually a GP. To see how that works, we look at the total weight. Now, these are variables. By the way, what kind of function of W and H is that? Again, outside the context of GP, if someone walked up to you on the street, what kind of function of W and H is that? It is a function of W and H, and I wanna know what kind it is. Linear? No. It's linear in W if H is fixed. And it's linear in H, so this is not – well, maybe I wasn't clear. There wasn't two questions, so I'll say it again correctly. What kind of function is this of left paren W comma H right paren? What's the answer?

Student:[Inaudible].

Instructor (Stephen Boyd):It's quadratic. Okay. Is it convex quadratic? Hey, wait a minute here. You should not hesitate.

Student: Yeah.

Instructor (Stephen Boyd):No, it's not. It couldn't possibly be. What kind of quadratic form does this have?

Student:[Inaudible].

Instructor (Stephen Boyd):There's nothing on the diagonal. Diagonals would be things like WI squared, HI squared. There's nothing on the diagonal. This is block like zero diag diag zero. Now, do you really have to think whether a matrix like that is positive semi-definite? I don't think so. Positive semi-definite things need non-negative on the diagonals. Of course, this has zero on the diagonal, but there's a rule. If you have a positive semi-definite matrix and it's zero on the diagonal, that row and column is zero. So this has split eigenvalues. So this is a quadratic function of W and H, but it is not remotely convex. It's got split eigenvalues – or concave. However, it's a posynomial because each of these is a monomial, and it's a sum with positively weighted blah blah – it's a posynomial. So that part is cool. Now the aspect ratio and inverse aspect ratio are monomials, so if I set them less than a number, I could divide by it, and I'd get a monomial – well, they're posynomials, so I can make an inequality of them. That's fine.

And the maximum stress is given by this, and that's a monomial here. Now the vertical deflection and slope are very – it's quite complicated to actually work out how much this thing – the tip deflects as a function of the width and height of each of these beam segments. It's very complicated. Okay? But it's given by a recursion, and the recursion looks like this. It says you take VI, and that's 12 times I minus a half here. I think this – in this case – this is for I equals one to four – no, it says for I equals one to N. Sorry. Everything's cool here. The VI is this thing. It's F over – that's a constant – E is Young's modulus, WI – these are the variables, so this term – hopefully, I can get this right. Yes. Okay. What is that term in GP terminology? What's that?

Student: Monomial [inaudible].

Instructor (Stephen Boyd):Monomial. That's a monomial, right? It's got a positive coefficient in front because I is one is the smallest thing here, right? F is positive. It's a positive force on the tip. Young's modulus, positive, and then it's actually W to the minus one, H to the minus three – HI to the minus three. Okay? So that's a monomial. And now you add this one that's gonna go from the tip back plus this. So actually recursively here, the first one is a monomial that – sorry, the last one is a monomial, and going backwards you take a monomial and you add it to the previous one. So by recursion, the VIs are monomials – sorry, posynomials. They're all posynomials. They're sums of posynomials all the way to the front. Then you get the YIs this way. Well, each YI is the next one – well, in the first one it's gonna be this plus that. That's in fact in the last case it's actually a monomial, so the last YI's a monomial. And the second one is posynomial, and all the others are posynomial. So what that says is that this is quite – these are very complicated expressions. You wouldn't want to see them written out, but they are indeed posynomials, all of them. So that's actually quite interesting. It's not remotely obvious. And let me say some of the things that it implies. That's not obvious, and I really doubt – actually, I think I can say with extremely high probability there's absolutely no one in mechanical engineering including someone with super good lots long design history stuff like that would know this. And let me just say one of the things you can conclude right away.

It says that if you have two designs, two cantilever beam designs – so imagine two. One is quite wide and high, and it makes no difference what they are. I'm gonna form a compromised design from the two beams. However, what I know to do is the following. I know I should take a log of everything. So the compromised design, instead of taking like WI and WI bar, and taking the average of the two – actually, my compromised design is gonna be the geometric mean because I know I really should be working – the problem mathematically is more natural in the log of the variables. So if you have one log width, you have another, I should take the average of the log widths and then take the X of that. That's the geometric mean. So the first thing it says is the way you should blend two cantilever beam designs is by geometric mean. By the way, that fact alone is not obvious at all.

And then there's a stunning conclusion about it. It would be this. If we're solving this problem – if you have two designs that meet all these specifications – these are fantastically complicated. When you start talking about the tip deflection and stuff like that – two designs that do it. If you take those two designs and form a compromised, a blended design given by the geometric means of the other ones, I can say something about the total weight of that blended design. I can say that it's less than or equal to the geometric means of the weights of the two designs. Okay? So for example, if those were two different designs that satisfied the constraints and had the same weight, that blend could only be better than either of them. Everybody see what I'm saying here? By the way, in other areas I've always tried this. I go to a field. I find someone who alleges to be a good designer with lots of experience and stuff, and I talk them through what the implication of geometric programming is in their field. And in only one case – it was Mark Horowitz who actually said yeah, that makes sense. He didn't say it right away either, by the way. Actually, I believe him. This is in the context of circuit design. I really believe him. He says now that makes sense. That's how you blend two circuit designs and so on.

I think there might be someone who does enough mechanical engineering that this would at least not sound stupid. Actually, it never sounds stupid. The question is whether or not – because you could make very strong statements about what happens when you make these blended designs. That's just one of the implications of what it means to be a GP. So how do you write this as a GP? Well, that's a posynomial directly, and you write out all the inequalities, which I won't write out. I won't go through this. I think you can go through this yourself. Let me look at one more example of a GP. It's a more generic one, also very interesting example. I don't know any practical uses of it, but there might be some. Actually, I've been itching to find one for years, never have. Here it is. Let's take a matrix that's element-wise positive, so a square matrix element-wise positive. So you might know about something called Perron-Frobenius theory if you've had a class on Markov chains or something like that, but I'll tell you what it is basically.

It's this. It says that the eigenvalue of A – A's not symmetric, so it can have complex eigenvalues. But it says that the complex value of largest magnitude – that it's spectral radius is that magnitude – is positive, No. 1, and also the associated left and right eigenvectors can be chosen to be positive. So that's Perron-Frobenius theory. I wanna see how many people have seen this in some context. I'm just sort of curious. That's all? Markov chains. Somebody took a class on Markov chains, right? What'd they say about that? Really? They didn't mention this? Okay. How do you know that the steady state equilibrium distribution is positive? Or in this case positive. This is the strong form. Where did you hear about it?

Student:A Markov chain class.

Instructor (Stephen Boyd):A Markov chain class? Here? No. It's not looking good for Stanford, is it? Where was that?

Student:[Inaudible].

Instructor (Stephen Boyd):University of Washington. Oh, okay. All right. And this is called the Perron-Frobenius eigenvalue. It's positive. And of course, what it does is it tells you the asymptotic growth rate of A to the K. So if you take powers of the matrix, if it's a dynamical system with positive entries, it says it'll tell you the growth rate. So for example, I mean this would come up in lots of dynamical cases. This would be for example an economy. It would tell you how an economy – or it might tell you how a multispecies system would work. Actually, the condition that A should be element-wise non-negative means that everything is I guess you would call it excitatory and nothing is inhibitory. I don't know if those are words, but you know what I mean, right?

So it means you start with a positive X. These are [inaudible] concentrations of something, and the concentration of anything does not inhibit the growth of anything else, so it would just lead to it. It would increase it. It would be excitatory. So in that case, A to the K tells you how this dynamical system would propagate, and the asymptotic growth rate, which could be negative by the way, is simply completely determined by this number. If this number is 1.05, it says that by the time you do this 100 steps, you've grown like 1.05 to the 100. If it's 0.96, it asymptotically decays as 0.96 to the K. So that's what this is. Fantastically complicated function of A, right? Because you calculate the eigenvalues of A, which is to mean you form the characteristic polynomial, which as a function of the entries of A you don't wanna see written out because it has something like N factorial terms. That's No. 1. You're gonna have to write – you're gonna have to find the roots of that polynomial, which again you can't do if N is 5 or bigger because there is no formula for the roots of a polynomial of degree 5 or bigger.

And then, once you get these N roots, you wanna take the absolute value and take the maximum of them. And I think by now you should be convinced, this is a fantastically complicated function of the matrix A. Okay. Well, another way to characterize this is that the Perron-Frobenius eigenvalue of a matrix is it's the smallest lambda for which AV is less than or equal to lambda. By the way, for some positive V – it turns out here the first thing you actually end up showing is it's that. It turns out you never get inequality here. It's always equality here, which of course makes sense because now lambda's an eigenvalue like that. But what's amazing is this – is you can now minimize the Perron-Frobenius eigenvalue of A where the entries of A are themselves posynomials by making a problem that looks like this. You minimize lambda subject to A of X sum – and these are all monomials in the variables, which are X, V, and lambda, and these are posynomial inequalities. So basically, log sum X comes out of this. That's how that works.

And you can make up some fake applications of this, I suppose. I think we attempted one in the book or something like that, but you can – it'd be nice to actually find a real one somewhere. That's a challenge, by the way. So I think what we'll do is we'll quit here, but let me say just a few things about where we are. We're sort of marching through all these problems. You will get to solve in a theoretical and in a numerical and practical sense every type of problem we're talking about multiple times. So we'll be doing all this for the next seven weeks actually, we'll be doing this stuff. So this is just your first introduction.

[End of Audio]

Duration: 73 minutes

ConvexOptimizationI-Lecture07

Instructor (Stephen Boyd):Let me start with a couple of announcements. Not announcements. Actually, one was we're interpreting relative silence from the class as an indication that for example, CVX is actually installing correctly and that everything is working. Now, there is another interpretation and that is that no one has started homework three yet. So if you find anything, any inconsistencies in anything, let us know. We already had a couple of typos in the user guide pointed out, but if you actually find errors otherwise just feel free to let us know. Actually, it's a good way to get it all debugged very well. Now, I have a very strange question for you. The question is this. It sort of came up in office hours and then I talked with the TA's about it. Now, hear me out before you laugh or start throwing things or anything like that. The question is this. We think we know the answer. The question is this. Would you like to have a midterm?

Student: No.

Instructor (Stephen Boyd):Okay. Okay. Actually, it's an honest offer. And let me just say what it is so this is just if you want and then you can say no again, but then at least I'm saying something very specific so here it is. Today, we're gonna start basically the last theoretical, I mean, the last sort of theoretical concept in the class, so that's duality. There's a whole section on algorithms and things like that, we'll get to that later. But then there's just gonna be a lot of applications and things like that and just doing this stuff. So in a week or two we'd be in a natural place where sort of the core theory is basically that it's all done. That's it. It's done. It's more of the same, a lot of applications, things like that. And so the idea is we could have – I'm just saying we could, this is an offer, we could do something like have a boring conventional style midterm for an hour and half somewhere and it would be these boring problems like which of the following functions are convex and all that stuff. But then the final could be – which will naturally be a mixture of – it'll have a handful of these things, but then mostly it'll be on problem formulation and things like that. What do you think?

Student: No midterm.

Instructor (Stephen Boyd):All right. All right. That's fine. The TAs predicted that by the way. Okay. All right. All right. That's fine. All right. Well, let's continue. The next topic is generalizing the quality constraints. By the way, as I said, I know we're going quite fast over this material. We're going at about twice the speed I think we should be going, but that's fine because things like the generalizing the quality – we're going to be using these in applications basically from now on. So you're gonna weeks and weeks of seeing real problems that use these things so if it feels like we're going too fast right now we are, but you'll have four weeks in the context of applications to absorb all these things. So generalizing equality constraints. So here we're gonna generalize the inequalities, by the way, later, we're gonna generalize the objective. That's actually, in some ways, more interesting or it's a more significant generalization. But right now, we're gonna generalize the inequalities to be vector inequalities. So here, FI of X until now has returned to scaler and it was FI of X is less than or equal to zero, we're gonna have FI of X return a vector, that's a vector inequality and it's defined by this cone KI. So in this case if FI is KI convex so it's a convex function with respect to this cone, this generalize and equality, then this is called convex problem with generalize and equality constraints. And you can generalize almost everything you've seen to this. For example, the set of optimal points for this problem is convex and all these kinds of things. Everything just goes through here with this. But there's a special case that's gonna be more important than the general one, and it turns out it's interesting. It's a very special case, but it turns out to be, in some sense, also the most general case and that's this. It's a so called conic form problem and you'll hear a lot about this. I don't think it's that interesting and I'll tell you why in a minute. Let me show you what is it. You'll hear a whole lot about it.

It's a very special case. Everything is affine so it's equivalent of a linear program basically. A normal optimization problem, if every function is affine you have a linear program because you minimize and affine function, subject to affine functions less than zero because you're conventionally called linear and inequalities. Affine equality, that's just conventionally called linear equality constraints. So here, everything is affine. You minimize a linear function. Now this is a very, very interesting inequality. This is affine and this just says that FX + G is less than or equal to zero with respect to this cone. That's all. So this is called a conic problem, a cone problem, something like that. You can type this into Google, a cone problem, and you'll get tons of stuff. I think it's interesting because it sort of generalizes the idea of linear programming to cone programming. If this inequality is with respect to the non-negative orphaned, so recall that if we erase this, that's the default. The default inequality between vectors with respect to R + and it means component wise and inequality. That's a general LP, that's a general linear program. So cone programming, I think it's caught on for several reasons. It is actually interesting to look at it, but it's caught on for a lot of reasons. First of all, if you're trained in traditional operations research you grow up – by the way, which is not me just so you know – but if you are trained or subjected to, maybe that's another verb one could use, a traditional training, everything you'll hear is about linear programming. The nice thing about cone programming is all you do is you drop this little K down there. By generalizing the cone from the non-negative orphaned you now have something that looks like linear programming so if that's your training and background it's deeply familiar, you've had three classes on it and now all you do is you generalize this and it means that everything is gonna be familiar and so on and that's what happens. A lot of things are familiar.

That's the first reason I think it's caught on. But the other is also interesting and you have to watch out for it, it's this, it's caught on because in fact the most highly developed solvers – we're gonna go in some detail in this in the last third of the class – the most highly developed solvers for convex optimization problems actually are conic solvers. So people who work on solvers naturally think that cone programming, conic programming and things like that is this huge important topic that everyone would be interested in. Actually, like linear programming, it's not. Okay. It's actually not. It's very important that if you worked on these things, fine, but for people who use these things, it's not that important. It's important in the following sense. There are a lot of interesting problems people really want to solve that transform to conic problems, and then, all the people who worked on the cone solvers can use their super fancy primal dual homogenous methods to solve them, but in some sense, that should not be the business of the general public. That should be the business of this small group of people who worked on cone solvers. I'm saying this by the way because there's a lot of people who worked on cone solvers who will see this tape and I just wanted to insult them, but that's all right. I hope it worked. I'm sure it did actually. All right. So this is a conic form problem and in fact, the one you will see most – there's a question?

Student:I'm wondering why doesn't I [inaudible] on the K in the first – is it because you can do different cones in the same set of constraints?

Instructor (Stephen Boyd):Yes. So the question was why the I in the K and the conjecture was it's because you can have different cones reach constraint.

Student: You can generalize it to make it –

Instructor (Stephen Boyd):And my answer was yes.

Student: It seems like you could generalize it all to one cone, the highest order cone or something like that?

Instructor (Stephen Boyd):Yeah, you can do all sorts of things. You could lump these all together and make it one inequality with one cone, which is the direct product of all the cones. So you could do lots of things. Okay. Now, in conic program what you'll see the most is something called semi-definite programming. And this is actually interesting, well, it depends, what you'll actually see is the combination of two types of cone programming which you'll see in a minute, which I'll talk about in a minute. So semi-definite programming, this is relatively new. So this is maybe 15 years old, but like everything else can be traced back into the 60s although people had no idea how to solve SDPs. So as a specific example I worked on SDPs before, in fact, the name had been fixed in the context of control and things like that and we were very, very happy when we could solve tiny small ones in an hour. Boy were we happy. But it didn't occur to us that you could solve them a thousand times faster and things like that and that the current state now would've been just unimaginable 15 years ago. This has sort of made the transition in the last few years from this kind of esoteric thing that a few people in research worked on to just kind of it being all over the place. It's very fashionable. It's also used commercially I should add in areas of finance and stuff like that it's kind of widely used. So semi-definite programming or SDP is very, very simple. It looks like a linear program except for one generalization. So we minimize linear function, linear equality constraints – that's so far linear programming. Here's the difference. There's an inequality and it's a matrix inequality. These are symmetric matrix's here. So this thing here is called an LMI, or linear matrix inequality. So that's what this inequality is. It's quite sophisticated. A matrix inequality is very complicated. To say that a matrix is positive semi-definite or negative semi-definite here is actually a very sophisticated inequality so from that point of view, they're right, it is sophisticated but sophistication doesn't come because you have a name attached to it or anything.

So this is an LMI and that's it. There's an interesting thing. Here as drawn there's only one LMI, but of course, you could have multiple LMIs, but the fact is that if you have a bunch of LMIs like this, if you have two, you can lump it together into one and you lump it together into one by simply making a block diagonal matrix here with the LMIs because if I have a block matrix and I say that the block matrix is negative semi-definite, it just means basically each block is negative semi-definite. So that's why we can, without lose of generality, work with just one LMI in general. Okay. Now, semi-definite programming is a generalization. Actually, everything you've seen so far except geometric programming, so let's see how. So it's a proper generalization of linear programming, so how do you represent this linear program this way. This inequality here, that, this one here is a vector inequality so this means element wise. Okay. How do you write that? Well, it's kind of stupid. If you take diag, this is the diagonal matrix with this vector on the diagonal, a diagonal matrix is negative semi-definite if and only if all the entries are non-positive. These are interesting for lots of reasons. The first is just sort of theoretical. It's nice to know that you have a problem class that subsumes other common problem classes. It's nice to know that in fact you didn't really have to know about SOCP quadratic programming. Is there a question?

Student:[Inaudible] problem solver or with a [inaudible]

Instructor (Stephen Boyd):That's a great question. So it depends. If the SDP solver – we're gonna talk about solvers later in the class, it depends on how sophisticated the solver is. If any modern SDP solver – if it handles sparsity and it's a – let's look at AX minus B here, diag minus B and realizes it's diagonal, it will have no loss compared to solving it as a linear program. So that leads me to my second point. SDP embeddings actually also have substantial practical use because if you write an SDP embedding of a problem it means that that other problem class can be represented in an SDP and it means that one SDP solver can do everything for you. Now, very roughly speaking that's what CVX does.

Student:[Inaudible] use that in their problem solver or [inaudible]

Instructor (Stephen Boyd):Nope.

Student: Is it because the [inaudible]

Instructor (Stephen Boyd):Absolutely not. Because it's gonna be a mixture of LMIs, all these constraints and it cannot call an LP solver, does not call an LP solver, will not call. So it's absolutely not the case. For example, in CVX or any of these other tools if you write something that happens to be an LP, it doesn't not call a different solver. It calls exactly the same solver as if you'd written an SDP. Okay. So no difference. So the point there is an SDP embedding actually then has a practical use and it's so expressive. SDP is so expressive that it can be used to represent a huge number of constraints and functions and things like that. We'll talk more about that later in the class. I should say also that it's come up now in theoretical computer science. I think it's in lots of area. Actually, I think we just found out that SDP traces back with a different name into some calculations in physics and chemistry in the 80s or something like that and it goes back into control in the 60s and things like that, so it's got a long history, but that long history didn't really particularly come with good computational [inaudible] so it was just theory. That's changed. SDP is now not just theory as I guess you all well know by the time you finished homework three, you will have solved, whether you know it or not a lot of SDPs. We'll get to that later. Okay. Here's a more interesting embedding. SOCP, so second order cone programming, the question is how do you represent this as a linear matrix inequality? Now, this is highly non linear and it's non differentiable and the answer is this and we'll see more about this later and in fact we have something queued for you, not on homework four, but on the next one, I think. This is a matrix here, which has a block – this one one block of the matrix is a scaler times the identity. Now, actually I don't know how many people remember about sure compliments, but let's just say we the teaching staff will remind you about sure compliments soon enough, but a block matrix is positive semi-definite. Now, this is rough so this is not quite right. It's positive definite if and only if the diagonal entities are positive and if something called the sure compliment is positive and the sure compliment is this minus this times the inverse of that times that is bigger or equal to zero. Now, for this particular matrix, that says CI transposed X + DI like this – let me put these together. Maybe if you do it on the pad here and capture both the big LM on this one and that one. There we go. If you can just capture this and what I'm writing here. It's this must be bigger than or equal to this thing AI X + BI transposed times the inverse of this guy, well, that's I inverse so I'll just leave as I, sure, I'll just put I inverse here, why not, and then that's a scaler so I'll put that down here. I'm using a loose parson here.

And then times AX AIX + BI and that is that term. Okay. Everybody see this inequality? So this one has to be positive because it's the last diagonal entry in a positive semi-definite matrix. It can't be negative. That's for sure. So this one I could square and put it over here. Okay. Or I can multiply both sides of this inequality by that. I can get rid of all of that. I get this. Now, I take the square root. When I take the square root, I have to be very careful. I can only take the square root here if I know CI transposed X + DI is bigger or equal than zero, but it is and I'll get exactly this second order cone constrained so this is an example of how this works. That's sort of the idea here. Okay. Let's look at some other problems. Here's one. It's eigenvalue minimizations. So here, I have a matrix which is an affine function of a vector X, so an affine matrix valued function looks like this. It's a constant matrix, these are all symmetric, plus and then a linear combination of symmetric matrix's here and I want to minimize the maximum eigenvalue of that matrix. Now, let me just point something out. That's a really complicated problem, right, because you take this matrix – so far, it's fine, every entry of this matrix is an affine function of X, but now to get the maximum eigenvalue value you form the characteristic polynomial. That's it. It's huge so you don't want to see it. You can't see it for a 15 X 15 matrix. Okay. You don't want to see it. You get some huge polynomial of degree N if N is bigger than five, there's no formula for the roots at that polynomial then dispute the fact you have no formula for the roots, you now have to select the largest of them. Okay. Now, that can switch. As you mess with a matrix the second largest eigenvalue value and the largest one can kind of go like this and when they switch this function is non-differentiable. Okay. So I just want to impress upon how complicated this function is.

It's very complicated, this thing, however, we can write it's non differentiable, it's highly non linear and so on. Now, on the other hand, we can actually optimize the maximum eigenvalue very easily as an SDP. It's totally straight forward. It's just this. You minimize T subject to A of X is less than or equal to TI. Now, to write this in this full form, I guess, these things I'm not gonna do, but you write it this way. And this is indeed affine in the variables X and T. That's this thing. So people just write it this way. It's fine. So that's eigenvalue. So eigenvalue minimization is quite straight forward. This is something that 20 years ago no one knew that this was straight forward, no one, 10 years ago a few people knew and so on. So this is new. This is not LP. We're not talking LP anymore. We're not talking stuff from 1953 anymore. This is recent history. Let's look at another one. Matrix norm. Now, matrix norm, as you know is a very complicated function. It's the square root of the largest eigenvalue of A transposed A. Now, when you look at this you immediately know you got a problem here. Land of max, well, we know that's a convex function. We know that. Right. Not only that, we know it's sort of SDP representable because we just showed how to minimize the length. But the problem here is this essential non linearly. This is quadratic and in an SDP, which after all has the constraint of a linear matrix inequality, you got the word linear in it. You're not allowed to have products. This has products here of A with A. So you have XI XJ terms. That's not linear. So the question is how do you turn a non linear problem into a linear matrix inequality and again, it's an SDP embedded so you take a larger SDP, it's larger but it's actually gonna be now affine or linear in the variables. So let's see how that's done. It's done this way. And you have to study this matrix quite carefully. It's in the queue. We're getting it ready for you. You have this block matrix here and this is TI and you want to know when this matrix is positive semi-definite.

Well, it's basically you have to have TI positive, and the other condition is that TI is bigger than A of X transposed times the inverse of that. I'll just put divided by T because the inverse of I is I times that. Okay. And very important here – that is a matrix inequality. Okay. There. T is positive; it's non-negative so this T can go over here safely like so. Now, you take a look at that and you realize we're quite cool because T is non-negative I can write this as TI is bigger than – no, I'll leave it this way. This basically says that T squared times the identity is bigger than this matrix. In a matrix sense that means that all the eigenvalues of this matrix are less than T squared. Okay. That means the maximum eigenvalue of this matrix is less than T squared, but the maximum eigenvalue of A of X transposed A of X is the maximum singular value of A squared and you take square roots and you're back to this thing. Okay. So that's the embedding. You'll see more of this. I just wanted you to sort of see it now. Also, it's interesting to see you don't have to know the details yet, later you will, that if you're actually curious about what CVX is doing, after all you can look at all the source if you like and find out what's happening, but if you look deep enough in it and then ignore all the silly testing and cases around there, the essence is every function implemented in CVX is an LMI representation. That's not quite true. In fact, let me say a little bit about that. I don't mind lying, but let me just – I don't do it that often and when it's for a good cause it doesn't bother me at all, but I'll just make this a little more accurate. It's not a total lie. What cone solvers really do and it comes back to your question to be honest, so the question is if you go to Google and you type cone solver or something like that, you'll find all the basic ones, you'll find there's 20 or 30 by now, there's a commercial one actually that you can get in excel so it goes on and on. But they all kind of do the same thing. What they do is they solve a conic problem which is like [inaudible] note the period, C transposed X subject to – and there's many ways to write it, but I'll write it this way – I'm gonna make these lower case – something like that.

Okay. And you can put an inequality constraint here. That doesn't matter. So they solve problems like this. Now, here I have to interpret these – these are the Ks you can have. They're typically sort of R plus so that's linear inequalities, they can be second order cone or larents cones so they can be SOC of some size K or something like that and they can also be sort of linear matrix inequality of PSD cones of size K. So this is really what they look like. As to whether the linear inequalities and the second order of cone inequalities are then further dumped into matrix inequalities and solved, is actually a question of semantics and things like that. So we'll just leave that out. But this is in fact really what if someone says cone solver, this is what they mean. This is not that interesting to people who want to use these things. I mean, I don't know what this would be like. This would be like looking at a very low level LA pack code and looking at the full calling sequence. It's of great interest to people who write low level code. It's very important that it be done right. That smart people thought about it, implemented it. But not all of us need to see the 14 arguments that you call, you know, DGESL with or something like that. Most of us are perfectly okay with just kind of the back slash or something in between like in a python or something where you'd actually let somebody else deal with all the memory allocation and passing pointers and all that kind of stuff. So I think this sort of has that flavor of that cone solver. So now I've been honest about what an actual cone solver in practice is. Okay. All right. Let's move on. What we're gonna do now is we are gonna to generalize the objective to be a vector. So, so far, we left the objective as a scaler and we generalized the inequalities to be vectors and you get things like cone programming and all this kind of stuff. Now, we're into the objective and this is a different thing all together. This is a very different thing when you change the objective because now you actually have to redefine the semantics because first of all it doesn't even make any sense so the first thing you have to do is say what on earth does this even mean. What does it mean?

What is FO of X is a vector with two things? What does it mean to say minimize a vector? Okay. And you get all sorts of nonsense about this so it's actually important to get all the semantics right. So a convex vector optimization is just like this except FO of X is a vector and there's a cone associated with it and there's a cone associated with it. And let me just say what this might mean. Let's do the scaler case. Let me tell you the semantics of scaler objective optimization. It goes like this. If the semantics are extremely simple and it all comes down to the following. If you have three people come up with a proposed X – I'll tell you everything you need to know about the semantics of scaler optimization. I look at the first X and I check the constraints. If any constraint is violated in the slightest I dismiss that X and that person, right, it means it's completely unacceptable and they can't say things like it was almost feasible or something like that. They're dismissed. Okay. Now, the remaining two people both pass the feasibility test. They're Xs are both feasible. They need the constraints. They haven't been dismissed.

How do I judge between them? Well, I mean, it's extremely simple. I evaluate the objective function for those two points. Okay. If one is less than the other I say that's better and I dismiss the other one because here I have two potential points, each of which satisfies the hard constraints, this one is better on the objective. Now, if they're the same, then the semantics – actually, what is the semantics if they're the same?

Student:[Inaudible]

Instructor (Stephen Boyd):No, they're not both optimal. I didn't say they were optimal. The answer is as far as that problem is concerned, you actually don't even have a right to have a preference between one or the other. You don't. Okay. So if you look at that and you say, "Oh, no, I much prefer this solution," then your objective is not reflecting your interest and your utility function. If you like one more than the other, but the objectives are the same, you're modeling is not right. Everyone see what I'm saying here? So that's it. Now, the key to this – in the scaler case is easy because an objective comes out to a number. It comes out to, like, $17 a cost per unit, in which case, $16.38 is better than $17, period, if it's a cost. If it's a revenue it switches. Okay. So the point is that they're all comparable. Now, suppose they're vectors. Now they're vectors and as you know with vector inequalities they're not linear orders. Linear ordering is one in which any two elements can be compared. That's one of the nice things about linear. There are linear vector orderings. Did we assign that problem? I think maybe we didn't. Does anyone know a linear vector ordering? This is just for fun. What.

Student:[Inaudible]

Instructor (Stephen Boyd):Alphabetic or lexicographic. Someone else was gonna mention something. Same thing? Yeah. So lexicographic ordering on a vector says you look at the first entry, you get two vectors, you look at the first entry and if one of the entries is bigger than the other, done, it's higher in the ordering. If they're equal, you drop to the second and then you see who beats on the second and if they're equal, you go to the third. So that's lexicographic ordering and it's a total ordering or a linear ordering is the name. So if they're equal all the way through then the two vectors are the same and there wasn't really a question anyway, right? So that's lexicographic ordering and it's a total ordering or a linear ordering is the name for that. It means any two points can be compared. That's why dictionaries work and blah, blah. Okay. All right. Most orderings of interest are not total orderings, most vector orderings, of course, the ordinary ordering on R is a linear ordering, but most vector orderings are not. And in fact, that's exactly what makes it interesting. So for example, in matrixes for example when you're talking about confidence ellipsoids and you say which is better, this or that, this estimation method has this confidence ellipsoids and this one has that one, which is better?

Well, of course, it can happen that one of the ellipsoids fits in the other and in which case you can say I like that one better, but of course, you can have this thing where they're not comparable. Were neither ellipsoid contains the other and then you have to say something like algorithm actually is better at estimating [inaudible] or whatever than elevation. This one on the other hand is better at doing this than that. So that's the idea. This is just to get the rough idea here. Okay. So we actually have to figure out what is the semantics here. Well, the way this is done you have to look at the set of achievable objective values. Now, that is a subset of a vector space of dimension bigger than one. We're assuming that's the case here. You look at the set of achievable values. That's the set of objective values you can possibly get with a feasible X. And now you say the problem is very simple. You say the point is optimal if that achieved objective value is minimal in O. You say it's pareto optimal if it's a minimal value. Now, you probably don't remember these from long ago, but the ideas are this. A minimum point of a set is one where every other point in the set is comparable to it and larger. So this is a minimum point. If this is the set of achievable objectives, this is a minimum point. This is the only case, essentially, when you can say unambiguously this point is optimal. It's the only time you can actually say minimize something and say X solves this because no one could possible argument with it here in this case. Minimum means you're better than every other point. Minimal means no point is better than you. Now of course if in total ordering these are basically the same. They're the same. But the difference here is you can have points – so if you look at this point here, all of these things are honestly beat by this point.

They're honestly beat by this point because they're comparable to this and they beat the object period. That's this point. But these things over here are interesting. These over here are interesting so the things in the second and the fourth quadrants, they're not comparable to this point. They don't beat it. Any point over here would beat that point unambiguously and the whole point about being pareto optimal or minimal in this set O is that there are no such points. That's the definition of being pareto optimal. So you would say, in fact, if someone said, "Yeah, what about that point and that point," you would actually say that they're incomparable, that neither one is better than the other and you would say, in fact, that in discussing the two designs there's a tradeoff between the two. You can trade off one objective for the other. Okay. So this is the picture. Some of these ideas are kind of obvious and you've probably seen them before in other areas.

Student: That curve – a significant portion of that curve also is pareto optimal?

Instructor (Stephen Boyd):Yeah. You see this little thing in here?

Student: Yeah.

Instructor (Stephen Boyd):That's not pareto optimal. Okay. And actually, that point, although you couldn't possibly see it certainly at video resolution you can't see it, or can you? I don't know. It's somewhere right around one line. This point is open and this point is actually filled in so these are pareto optimal down here and all of these are. Right? Another name for pareto optimal is not stupid. That's another technical term because for example if someone says, "Yeah, well, that's my choice," okay, the technical name for that is stupid and the reason is this – if you go down here like this, you know, everything like this – anything in there is a point that unambiguously beats this. So it would be stupid to choose this point. Any of these other points in here would actually be comparable to that point and unambiguously better. So that's the other way to think of it is just not stupid. But the way to do it is very simple. It basically says minimal means you look up in here and you want everyone else to be up there. Those are the places that are unambiguously worse and pareto optimal says you look down and left. I mean, this is in this very simple case. You look down and left and these are all the places which are unambiguously better than you and you want to make sure that no one is unambiguously better than you. So that's it. And you've seen this idea many times – my guess is you have. Or even you've thought it and just didn't have a name for it or something. All right. Vector optimization comes up in essentially only two practical contexts. One is where the ordering is with respect to matrixes and that's where you're comparing and that actually comes up all the time. It comes up in experiment design; it comes up in estimators and things like that where basically the objective is something like a covariance matrix like an arrow covariance matrix. Everybody wants a small arrow covariance matrix. It defines an ellipsoid. Everyone wants a small confidence ellipsoid. That's clear. The problem is those are not totally ordered, right? And we'll talk about what happens in that case. So that's the first case when these are matrixes. That's sort of the subtle case. The other case is actually much simpler although it comes up maybe more frequently. It's just multi-criterion optimization and here the cone is nothing but the [inaudible]. So here basically it's called multi criterion optimization. Your F0 has got a bunch of components and people call this a multi-objective problem, that's another name for it and you will hear a lot of nonsense put up around all of this.

You hear people say, and of course, it's true, you hear people say things like, "Oh, all optimization problems are multi-criterion," well, like, duh, so what? I mean, that's completely obvious and we'll see that it relates immediately if you know what you're doing to ordinary optimization problems. So we'll get to that in a minute. In a multi-criterion problem, you have multiple objectives and if you want to think about something specific here you might think of an engineering design, let's say a circuit and one of these would be something like power, one would be delay, one would be area, I mean, just for example. Okay. Or this could be some control system or something like that and one of them could be, again, something like power, one could be how close you're tracking and the other could be something like the roughness of the ride. The point is there are multiple objectives all of which you want small. Once you think about it you realize everything has this form. In an estimation you'd say, "Well, I want something that matches my observed measurements well," I mean, it would be foolish not to have something like that, but at the same time, I want an image that's smooth, or I want one that is simple in some method. In some sense, those are all gonna be multi-criterion problems. Okay. And I'll say a little bit about how it all fits together in a minute. Okay. Roughly speaking, you want all of these small. Now, the one thing you can say is this, if someone comes up with a point and you beat someone else on all objectives – or I should be more precise, if you meet or beat them on all objectives and beat them on one, you're better. Everybody got that? This is like a core in economics you go through all of this. You'd see all these things. That's what it means to be better. So pareto optimal means no one beats you. That's pareto optimal. It means basically that if you have two pareto optimal things, you have to have traded something off.

They have to beat you on something; you have to beat them on something else. Okay. Now, pareto optimal, well it says exactly that. It says that if one thing is less than or equal to the other the same objective that's in a vector ordering then they're equal. And if you have different pareto optimal values there's a tradeoff between the objectives and the classic thing from lee squares or anything else would be something like that so you'd write it this way, you'd say minimize with respect to R + squared. I mean, this is a bit of a pedantic way to write it, but it is actually exactly what you mean. You say please minimize with respect to the second order [inaudible] the following: two objects, both by the way convex here. Norm X minus B squared, that's like a matching criterion. That's how well do you map something. But at the same time, please do it with a small x. Okay. How do you judge such a problem? Well, what you do is this. X by the way could be a vector in 10,000 dimensional space, but there's only two objectives, so in principal, what you do is you check every vector in our 10,000 so you plug in every possible X and you get a score, which is a pair of numbers, non-negative numbers and you put a dot at every single point. This is conceptual. You now do this for all 10,000 points and you get this shaded plot that looks like that. In fact, it's some kind of a hyperbole or something, it doesn't matter what it is, right? Now, also I should add that O has some interesting structure out here, but it's not interesting to us. What's interesting for us is down and to the left. That's the only thing interesting to us. And therefore, whatever O looks like up here is actually of no interest unless it kind of does some weird thing and wiggles around and comes out down here, but it doesn't, okay? So we're only interested in what's down and left. So in actuality the only thing that's interesting here is what's shaded here is the pareto optimal curve. Now, you saw this if you took 263. This is called the regulization and so on, so you've seen all this.

I'd say the technical term for this is quite stupid. Let me justify that. It's quite stupid because, first of all, there's better points. That's number one. Number two, it's not even achievable, so that's why it seems to me it's even worse than stupid, but anyway, this should be clear. Here's a generic example. It looks like this. I have two objectives and let me explain the problem here. I have a vector that represents an investment portfolio so XI, for example, might be the fraction – people set this is up differently, but a classical one is this – XI represents the fraction invested in asset I. So you have N assets, negative – if you want to have negative, I mean, here we've ruled it out, but the point is if you want to have negative it means you have a sure position. If that asset is cash, it means you're margining. You're borrowing money to finance the purpose of other assets or something like that. But here to make it simple we just made X a positive and you'd say that there are no short positions and if one of those assets is cash you'd say no borrowing. No margining. Okay? So that's the idea. That's the fraction. And of course as a fraction, it's got to add up to one. Okay. Now, what happens here is you invest in these assets as one period and what happens is there's a return or the prices change, so you can call P – it's a vector of relative asset price changes. So if you form P transpose X that gives you the total return on your portfolio. Okay. Now, obviously, if the returns were known – in fact, let's discuss that because it's a very short discussion. If I told you the returns, what's the best possible portfolio?

Student:[Inaudible]

Instructor (Stephen Boyd):What is it?

Student: You put it all into the best one.

Instructor (Stephen Boyd):Thank you. You put it all into the best one. So you find and someone tells you, reveals to you what the returns are, it's extremely simply, you simply put EK, where K is an index corresponding to whichever one had the highest return. Okay. That was the discussion of deterministic of finance. It was a short one. So the interesting part is this. It's a random variable. So the return is actually a random variable so it's a random variable here and you can get much more sophisticated, but the basic one goes like this. So P is a random variable and therefore P transposed X, that's the total investment return, is a random variable. Okay.

And now obviously you want it large. What does it mean to say a random variable is large? By the way, there's tons of ways to say a random variable is large. Okay. Lots. You could say it's 90th quintile is large, it's 10th quintile is large, you could say all sorts of things. There's many, many ways of doing this. The simplest thing that makes any sense at all is this, you evaluate the expected return and the variance, or you could take the square to the variance and get standard deviation. That's sort of the minimum that makes any sense at all would be this pair. And now you have a bi-objective problem because everyone wants high-expected return and everyone will take a lower risk over a higher one. Let's just say that people are risk adverse. So what that means now is you plot these things by two things; you'd look at the risk as measured by standard deviation of return – of course you can get much more sophisticated measures of risk like you can have downside risk, value at risk, you know, it goes on and on. But the simplest measure is just variance. Then you have all sorts of long discussions about this like somebody would make the stunning – are you ready for an amazing observation – how about this, somebody points out, "Yeah, we judge portfolios by risk, but it turns out – well, imagine what would happen if your return was much higher than you thought, would you be unhappy or happy?" And you'd say, "Well, let me think about that for a minute. I guess I'd be happy and they go see variance is the wrong measure of risk, you care about the downside variance." So these are all totally obvious things, but let's just – and by the way, if you make any assumption about the distribution, like, if it's galsium, then all these go away. So let's just assume variance is a measure of risk or standard deviation and what you'd do is you'd solve this problem – sorry, I haven't said yet how to solve a multi-criterion problem, but this would actually be the pareto optimal boundary. You'll do plenty of these, just for fun.

Now let me say something about what does it mean to solve a multi-criterion problem? Actually, that's an interesting question. We all know what it means to solve a scaler problem; what does it mean to solve a multi-criterion problem? Well, there's many definitions. One is this, if that multi-criterion problem has an actual optimal point then for sure solving means you better find it. So if there was actually – and it could happen. Here in a finance problem you could have something where basically you have one asset that beats all of the others in both risk and return, for example, in which case this would be a stupid problem. The pareto optimal curve would be a single point. Everything else would be down and to the right meaning that you can choose this portfolio or if you wish, there are many portfolios to choose from, but all of the have both lower return, higher risk and one of those inequalities is strict. So that's what it would mean in this case. It could happen. It obviously doesn't happen in practice, but that could happen. Okay. Now the other meaning is something like this. You could define the semantics of solving a multi-criterion problem as producing a pareto optimal point. That's another meaning for what it means to solve it or an interesting one would be to produce one or more – or actually you want a method for exploring the pareto optimal points. That's actually probably what you want when you say how do you solve it. So most methods for solving multi-objective problems will have some parameters or a joy stick in them that would allow you to move around on the pareto optimal surface, which would be called exploring the pareto optimal surface and that kind of thing. So classic methods, probably hundreds of years old, scalarization. How does that work? In the general case, actually it's quite elegant. It works like this. You choose a positive vector, positive in the dual cone, okay, in the dual cone. Now, you might remember this.

When we looked at it, it was completely context free and had no actual practical meaning so you probably ignored it, but one way to say that F0 of X is K convex – one condition for that is the following; it's that lambda transposed F0 of X should be convex. That's a scaler function. It's a convex function for all lambda incased [inaudible] in the dual cone. So that's one way to say it. In particular, this is a scaler convex optimization problem. Okay. By the way, this is already said something. In the case of multiple criterion we'll see it's something very simple. It's a positive way to sum up the objectives. I mean, everybody knows that. You want to minimize a bunch of things, you put positive weighted sums and you wiggle the weights around to change the tradeoff. That's obvious. It's much less obvious when we get to matrixes. You want to minimize, for example, estimation error covariance, it says take a positive definite matrix, you know, lambda and minimize trace lambda sigma. That's what it says. This is not obvious just to let you know. Okay. So here's what happens. This is a general problem, non-convex and here's what is really happening. So here in this problem if you take a lambda like this and you minimize this, it basically says you go as far as you can in this direction and you might find that point right there. That's pareto optimal. Okay. Now, by wiggling lambda you're changing the slope and it says for example you can get this point. Now, what's very interesting here is that this point right here is pareto optimal, but you cannot get it by any choice of lambda. It's shadowed. There's no way of getting it. If you want to wiggle around on the pareto optimal design, you change the weights and it's kind of obvious how you change the weights. Actually, the choice of weights relates to our very next topic which is duality, but for now, you can think of the weights as simply a joy stick that moves you along the pareto optimal surface so that's the right way to do it.

So if you go back to the risk return tradeoff now I can tell you how those plots were obtained. They were obtained as follows; we maximized this minus that and I varied lambda from very small to very large and I solved this QP for each value and that's how I got the curve. Every one of those is risk return pareto optimal and this is in fact gets you everyone in the convex case. By the way, this has a name; this is called the risk adjusted return. That's the average return and now you've penalized the return by some adjustment for risk and then you would say something about gamma is your risk aversion parameter because it tells you how much mean return are you willing to give up for variance in your total return so this would be a risk aversion parameter. For us, it's a weight, it's in the dual cone of R2, it's a weight that simply weights variance versus return. That's all it is. So that's that. Okay. So this finishes our whirlwind tour through optimization. You should have read chapter four by now and you should read of course the CVX users guide for that and they should be coherent in fact. Well, I hope they're coherent. If they're not coherent, let us know. We're by no means done, but the rest of the class, every homework, no exception, you will see problems where you will formulate them as convex problems, you'll do numerical solutions, you'll work out more theory as we do it. So in some sense, we've kind of finished our first coverage of problems – we do problems from now on period in applications and things like that. Now, we're going to enter a new topic, which is actually sort of our last sort of theoretically oriented topic. There will be some others later, but this is a very important one. It's duality. It relates exactly to these weights. And I'll start on it. This is in chapter five. You should start reading chapter five. Okay. Duality is a very interesting topic. It's interesting theoretically, it turns out it has extremely interesting computational implications. It also turns out it's extremely important in all practical cases so whenever you use optimization, ever, you have to be aware of duality. And by the way, it's not just a theoretical thing; however, if you randomly search in the literature on duality you'll imagine it's just something that people who do theory worry about. It's immensely practical and we'll see that into it. There's something I forgot. I pushed it on a stack and didn't come back to it. I want to come back and say something about multi-criterion optimization versus scaler. And I just wanted to put them all in the common parent so the common parent of scaler and multi-criterion optimization is this. If you want to put them on a common footing, you'd introduce a concept called the hardness of a constraint. Okay. So that would be the word used. A constraint that is infinitely hard or let's say an objective that's infinitely hard. The problem is constraint already means hard and objective already means soft so let's call them objectives, but let's call them functions.

There we go. Functions. F0, F1 and so on, we're gonna call them functions. Infinitely hard constraint – so you can do the concept of hardness by plotting your irritation versus the value of FI of X. So here's 0, that means you're not irritated at all and the semantics of the objective, your plot looks like that. That's your irritation; I guess this means you're pleased. Okay. And this says basically the following; it says that you can make F0 – F0 can be large, if it has to be large, it has to be large. It irritates you, but if it has to be large, it has to be large, so this would be a soft irritation function, right? So that's the picture there. You can also be pleased and if you have a choice between two points, one of which was sort of here and one was here, you'd prefer that because you're lower and it means you're less irritated. Everyone see this? Suppose you wanted to change this irritation function; what irritation functions work right away with convex optimization, just instantly? Let me draw an irritation function and you see if you like it. How about this? I'm down there. That's my irritation. And then someone says, "Why," and you go, "Not a problem. If this is the area, yes, I always like things to be smaller, always, but you know what, when things start getting bigger than some threshold, it starts irritating me super linearly," that's what this is. Can I express this as convex optimization problem? I can because I would simply take this function of F0 – this function is convex and it is increasing and therefore I can do that. Okay. Then the objective value would match yours. For a constraint, the canonical irritation function looks like this; it's along the axis so I kind of have to draw it that way. Okay. That's zero. That's your canonical irritation function. It basically says if – let's say in a classical problem, if F3 of X is less than or equal to zero, it doesn't irritate you at all. By the way, the same is also true in the semantics of the basic problem that a smaller value of F3 means nothing to you. Absolutely nothing. By the way, you might find someone in practice say something like "Oh, no, I prefer a smaller – I prefer," but the actual semantics of the base optimization problem says that F3 of X equals 10 to the minus nine, right, and equal minus 100 are equal for you. This is by no means any better than that. They are both feasible. End of story. That is the same as that. This and this are also equally and infinitely irritating. They are totally unacceptable, and here you can't say come on, I just barely, you know – now, as a practical matter, you might argue and if you look carefully, what numerical solvers will return, I mean, obviously, they'll return things that, for round off reasons, might violate things slightly.

Anyone who asserts that there's any difference between these means that you haven't formulated the problem. The common parent is the following is that you look – this is sort of a very general thing where every constraint has a certain softness and hardness and if you're willing to label all the merit functions except for one as absolutely hard and one as soft, then you have a classical problem. That never really happens. If you label them all as soft, you'd have a multi-criterion one. What really happens is that some constraints are non-negotiable and others maybe are and they might be negotiable for different ways. And you might even have weird things like you could say, "Well, you see this power constraint or whatever," but actually if the power goes up to two watts everything is fine because I can use this – I mean, actually, I like less power, but I can go up to two watts with this package. If you go above two watts, boy that's not good. Let's get back to duality. Maybe I'll just say a little bit about it. It's Lagrange duality, which we'll see it in a bit. I'll just say a little bit about what it's used for and all that kind of stuff. Hopefully, this is will de-mystify the often mystical Lagrange multipliers, which is generally taught as a behavior in an advanced calculus class. So this will completely de-mystify it. That'd be the first thing. Actually, it's got a lot of interesting things. In fact, what it's gonna do is it's gonna end up producing – from any optimization problem it will produce a convex problem, which is called the Lagrange Dual. Okay. If you start with a non-convex problem, you'll get a Lagrange dual, which is convex. We can solve convex problems, I mean, roughly speaking. We can solve them. Okay. That Lagrange dual problem is actually gonna give a bound on the optimal value of that original problem. Okay. If that original problem is convex, it's gonna be a sharp bound.

There's some conditions, but it's gonna be a sharp bound and it's gonna be very interesting because it's gonna give you alternative methods for solving that problem. It's gonna give you all sorts of interesting things. It's also gonna be extremely interesting when the original problem is non-convex. Well, for the Lagrange dual, it's gonna be a convex problem, we can solve it. And you're gonna get bounds on that original problem. We'll see a few cases where that actually happens. So that's a case where a problem that looked hard by forming a Lagrange dual and then doing some extra math to show the bound is sharp, actually is solved by convex optimization. By the way, there's gonna be lots of those. Okay. Any time someone says a problem is easy and it's not posed as a convex problem, it's likely that in fact it's because the Lagrange dual is sharp and that's true for many [inaudible] problems and all sorts of things. The other thing that could happen is you could get a bound on a problem not known to be hard, but suspected to be hard and there's amazing things that people have recently proved. There will be a semi-definite programming Lagrange dual of a hard [inaudible] problem and they prove things like specific ones, like, Maxcap for example, when you solve this Lagrange Dual the bound is never more than 13 percent off. Crazy things like this. That gets the complexity theorists all excited and indeed that result got them very, very excited. As you can imagine, right, but also as a practical method for generating sub optimal solutions of that problem, these things can also be fantastically useful. So that's our next topic. It's gonna be essentially the last sort of core theoretical topic in the class and I guess we'll start in on that next time.

[End of Audio]

Duration: 77 minutes

ConvexOptimizationI-Lecture08

Instructor (Stephen Boyd):We'll start with an announcement. It should be kind of obvious anyway. You should start reading Chapter 5. So I'll go fast, not that fast, on our next topic, but you should be reading if you want all the details and things. You should be reading along with or even maybe ahead of us, Chapter 5. Okay. So we're really gonna start this topic of duality. I think last time I did nothing but say a few things about it. They were kind of incoherent, but maybe I'll make them coherent today. One way to think about duality is it's a way to handle the constraints by incorporating them into the objective. That's the basic idea. So let's see how that works and it fits in exactly with what I was talking about last time, which had to do with this concept of irritation versus value for a function and the idea of a hard constraint where you go from completely neutral to infinite irritation as opposed to a soft, something like an objective, and an objective, it's just a linear irritation function or something like that that means the larger the function is the more irritated you get and the smaller it is, the happier you get. That's gonna tie into the idea of duality. All right. So first we start with a couple of definitions. We start with a problem like this so minimize the objectives, some inequality constraints and quality constraints. We are not going to assume this is convex and in fact, some of the most interesting applications of the material we're gonna talk about now, occur when this problem is not only not convex, but it's a problem known to be hard. So that's gonna be some of the most interesting applications. We form a lagrangian and the lagrangian is simply this, I mean, it's really kind of dumb. You take the objective and to the objective, you add a linear combination of the constraint functions and the equality functions.

So that's it. Here you say, for example, that lambda three is the lagrange multiplier or dual variable or price, I'll justify that term soon, associated with the third inequality, so FI of X less than zero, that's what lambda three is. And in fact, you can even sort of get a hint at what's going on here. If, for example, lambda three were six that would mean the following: up here this says that if F3 is less than or equal to zero, it's perfectly okay; if it's bigger than zero, it's infinitely bad. When you replace this constraint by this charge, lambda becomes the price or you can call it a penalty and it basically says FI can be positive, that's not a problem now, but you're gonna pay for it, and for every unit that FI is positive, you're gonna pay six here, whatever those units are in. Now, the flipside is actually quite interesting. You actually get a subsidiary for coming in under budget. Up here, as we said last time, there's absolutely no difference between F3 of X, for example, being minus .001 and minus 100, absolutely no difference, both are feasible and you have no right to say that you like one better than another because it's not an objective, it's a constraint. Now, when you convert that one constraint to this sort of thing, something interesting happens. When F3 is less than zero, you actually get a subsidy because it's 6 times whatever margin you have, so when you convert this constraint to a term in a lagrangian, you actually get a subsidy for coming in under budget and you do like F3 equals minus 100 more than you like it minus .0001. You get a subsidy. I'm just hinting at what you're gonna see soon. That's a lagrangian. Then we look at the so-called dual function or the lagrange dual function, it's got other names, but let's look at it. So the lagrange dual function is actually very, very simple. It just says minimize this lagrangian over all Xs. That's what it says. So you just minimize the lagrangian. Now, actually it's very interesting because what it's really saying is this, and by the way, it's a function now of these prices or dual variables. All sorts of ways to interpret this. You can interpret this as sort of the free market approach or something like that.

This is the constrained approach where you simply say by fiet FI of X has to be less than zero, HI of X has to be zero then you could say, no, you know what, we're gonna let FI float above zero, no problem, we'll charge you for it the lambda I's are positive. That would be the meaning here. But if FI goes less than zero, we'll actually pay you for it. You'll be subsidized and this is sort of the optimal cost under these sort of market prices. All of this will become clearer as we move on with this. Now, here's an interesting fact. If you look at this function as a function of lambda and nu, what kind of function is this as a function of lambda and nu for each X. It's affine. Yeah. It's linear plus that's a constant. Okay. So it's affine. And the [inaudible] of a family of affine functions is of course concave. So this function G, you can think of it as the optimal – we'll get to it later – as a function of the prices, even if the original problem – even if these things are not convex, sorry, these are affine, that's not convex. This dual function is absolutely concave so that's – all right. Now, we get to something very simple, but it's one of those things where you get a sequence of 12 simple things and you know the right sequence of 12 simple things will lead you to a very interesting thing. So trust me, that's what we're doing here. It looks very complicated, it's quite profound. It says, basically, if you can evaluate G as a dual function, you're gonna get a lower bound on the optimal value of the original problem. That's what it says. So the argument is just embarrassingly simple. Look at this thing and imagine X is feasible. For any feasible X FI of X is less than or equal to zero, but if lambda I is bigger or equal this term is less than or equal to zero so therefore, this whole thing is less than zero. If you're feasible, HI of X is zero so it doesn't even matter what the side of new I, this is zero and, therefore, it says that the lagrangian is less than F0 of X or any feasible X.

Now, for infeasible Xs that's false, but for feasible Xs it's absolutely true that L of X, lambda nu is less than or equal to F0 of X because that's zero and that's less than or equal to zero. By the way, note the level of the mathematics being employed here. It's quite deep. It relies on the very deep fact that the product of a non-positive number and a non-negative number is non-positive, and also that you can add such numbers up and you still have something less than or equal to zero. So I just want to point out nothing has been done here. It's just embarrassingly simple. Okay. Now, if you then [inaudible] this over X well then obviously it's less than F0 of X tilde. This is true for any feasible X tilde and there's no conclusion possible other than this. Okay. So let's look at some examples. Let's do least norm solution of linear equations, so here's – now, of course this is stupid. We know how to solve this problem analytically, you know how to, it doesn't even matter what the solution is. It doesn't matter. We know everything there is to know about this, but just as an example let's see how this works. Well, the lagrangian function is this; 2X transpose X, that's the objective, we add new transpose AX minus B so here, by the way, you have to write this as AX minus B = zero. You have to decide what H is. H is either AX minus B or something that's B minus AX so all that would happen there is the sign on nu would flip or something, but it wouldn't matter. I've written it this way, AX minus B = zero so. We're gonna minimize this over X, that's completely trivial because that's a convex quadratic, that's affine in X and you just take the gradient, you get 2X plus A transpose nu = zero and this is the optimal, that's the X that minimizes the lagrangian here. All right. Now, we take that X and we plug it back into here to get G so when you plug this in here that's the X that minimizes the lagrangian, you get this and you get some – well, first of all, let's take a look at it.

Evidently, this function here is concave because that is a positive semi-definite quadratic form. All we care about is a positive semi-definite quadratic form, but it happens to be positive definite. No, actually it doesn't matter in this case because I'm making no assumption about A whatsoever in this case; everything is true no matter what I assume about A. Okay. So this whole function is concave quadratic so there it is, which we knew had to happen because the dual function is always concave. Now, here's what's interesting. And we've already learned something. It's not a big deal because I know how to solve this problem, but look at this. What it says is the following; if you come up with any vector nu at all, which is I guess the size of B, the height of B, let's call that M, you come up with any vector nu at all and you simply evaluate this function then whatever number you get is a lower bound on the optimal value of this problem. In this case, it's totally useless. If that's a small problem – say, X has a thousand variables or something like that, this is goofy because we know how to solve this problem extremely efficiently, get the optimal solution, we don't need a lower bound on it. This is actually immediately useful. Let's look at a standard form LP. So I want to minimize C transpose X subject to AX = B X figured or = to zero. Just standard LP. Okay. The lagrangian is C transpose X plus, and I'm gonna add a lagrange multiplier for this, that's nu, transpose AX minus B and then here – to put this in standard form you really want to write it. You have to write it as minus X is less than zero so these are the FIs over here. Okay. So if I simply form this thing and that explains the minus sign here on the lambda. Okay. So that's your lagrangian. What kind of function is the lagrangian in X? Hey, look at that, that says linear doesn't it? And it seems to me that that is false, I believe. That's the name for something that's not true. Isn't that correct? You agree with me? The lagrangian is not linear in X. That's ridiculous. Is that affine? L is affine in X. It's affine because there's this constant term minus nu transpose B here. Okay. Now, this leads us to something. It's actually related to something on the homework from this week, which was – here's a very, very simple linear program. Ready for it? No constraints. How do you minimize an affine function? How do you minimize an affine function? More than a small number of people were stumped by this because it's such a stupid question, I think. Actually, it's not a stupid question, sorry, it's just that they wanted to make it more complicated. How do you minimize a linear function?

Student:

[Inaudible]

Instructor (Stephen Boyd):What's that?

Student:[Inaudible]

Instructor (Stephen Boyd):What's the minimum of a linear function? Just in R2 I have a linear – what's the minimum of X1 minus X3?

Student:[Inaudible]

Instructor (Stephen Boyd):X2. Sorry. Go ahead. What is it?

Student: It's minus infinity.

Instructor (Stephen Boyd):Of course. It's minus infinity. You have level curves which are hyper planes and you just go as far as you can in the direction minus C, it's minus infinity. So there's nothing, there's no mystery here and what's the minimum of an affine function. There's no mystery here. Okay. By the way, notice that that's a valid – so if G is minus infinity, it's cool, it's just minus infinity and note that it is indeed a lower bound, however it's an informative lower bound because if somebody comes up to you and says, "Can you help me get a lower bound on this," you can just automatically say minus infinity. Exactly one. What's that? When the function is zero. And that's it. That's the only way. So in fact, if you minimize this over X, you will get minus infinity. One exception. If C plus A transpose nu minus lambda is zero then this whole thing goes away and you get that. So the dual function for the standard form LP is a very interesting function we're going to look at it very closely. It's this. It is a function which is linear on this weird affine set and it's minus infinity otherwise. By the way, that function is concave. Right? You can just visualize it. I mean, it's kind of weird. It would be an R2 like my drawing a line like this and here are the values, 0, 1, 2, -1 and then you have to visualize this so if you want to make a graph coming up. Slope this line up and now what I want to do is everywhere off that line the function guy is minus infinity so just make it fall off a cliff everywhere else. That function is concave. Well, it has to be concave because we know the G is always concave. Okay. But if you want to visualize it, that's what it looks like so it's a weird thing, it's linear, but then off this thin affine set, it falls to minus infinity. Okay. So that's what G is. Is there a question?

Student:[Inaudible] LP is affine [inaudible]?

Instructor (Stephen Boyd):Okay. So actually the question is this function, which I write this way. You just have to blur your eyes because I'm asking you what kind of function of X is it? Okay. So let's look. That's is a constant. That is a constant vector, but well, if I include the transpose here, it's a constant roe vector, therefore, this is linear in X, I add a constant so it's affine. Does that make sense? Okay. Okay. Now, this is really interesting. We can actually say what the lower bound property is. So the lower bound is this. This function is the dual function. It is always the lower bound on that LP. Now, of course, if you randomly pick lambda and nu, you're gonna get minus infinity. Just minus infinity. Okay. In which case it's still a valid lower bound, but it's just a completely uninformative lower bound. It's the lower bound that works for all problems. It's the universal lower bound. So let's see what we've come up with. It says the following. If you have this linear program here and someone says what is the optimal value of it, well, it depends on the context. If a person has a specific A, B, and C, and actually just wants to know solve my problem, you can run some code and solve the problem if it's not too big and if that's what they're interested in. Okay. However, you can make a very general statement. You can say the following. If you find any vector nu by any means, it doesn't matter how you find it, it's no one's business how you found such a nu, if you found a vector nu such as A transpose nu plus C is a non-negative vector, then you evaluate minus B transpose nu, that's the lower bound on this LP. That strings together three or four totally obvious things, but I think you come up with some – that's not obvious. Okay. Let's do a quality constrain norm minimization. So here you minimize the norm of X subject to AX = B.

By the way, we've seen that – in fact, I guess we just did that problem two examples ago – not quite. We did the case where this was norm squared and where this was the two norm, now, it's completely general. Well, the lagrangian is the [inaudible] of norm X minus and then some horrible thing here, which is affine and here we have to be able to minimize norm X minus nu transpose AX, this is a constant so it's totally irrelevant and you have to be able to do that. Now, this goes back to the idea of a dual norm and let me go to that so let's look at that and if I want to minimize this thing – the question is what is this thing, what do you get here, right? And the answer is actually pretty [inaudible] straight from dual norms. In fact, we can do it for the two norm first just as a warm up. So for the two norm you'd say, well, look, if norm Y is bigger than one in two norm then I can align X with it in that direction and then this thing sort of over powers, it has enough gain to overpower this one and I can make it go to minus infinity. Okay. Now, on the other hand, if norm Y is less than or equal to one [inaudible] tells me that this thing is less than that and, therefore, this whole thing is bigger or equal to zero. So I could never ever make this thing negative. On the other hand, by choosing X equals zero, I can make it zero, so that's clearly the optimal.

So this is equal to and this generalizes now to a general norm. It's either equal to minus infinity if the dual norm of Y is bigger than one or at zero otherwise. Okay. And in fact, that's the dual norm. It's from the definition of the dual norm. So this is what you get. All right. So applying this up here gives us exactly this. This is our Y and here you have it so once again, this dual function is not totally obvious. I mean, it's not an obvious thing. It's something, it's linear, it's a linear function, but it's got a weird domain and in this case, it's the set of points nu where the dual norm of A transpose nu is less than or equal to one. Okay. That's that. Okay. And then you can go through the argument here and I won't go through it, but actually now you've got something totally non-obvious. It basically says if you can come up with a vector nu, for which A transposed nu is less than or equal to one in dual norm, then B transpose nu is a lower bound on the optimal value of this problem. Here's an example: ready for a dual feasible point? Nu equals zero. Well, let's check. That's zero. Zero is definitely less than one and now we have a drum roll to find out what lower bound we've come up with. The lower bound is zero and that's actually not particularly interesting here because the objective is zero. Okay. So that's what it says here. Okay. Okay. This gives you a parameterized lower bound. It's parameterized by nu. Okay. Now, we're gonna look at a problem and we'll see it a whole bunch of times. It's actually just a simple example, it's a perfectly good working example of a hard problem. It's two-way partitioning. It's embarrassingly simple. It goes like this. I want to minimize a quadratic form subject to XI squared equals one and this means XI is plus or minus one. Okay. And let me first just say a little bit about this problem. We're gonna see it a lot and just so you get a rough idea of what it means.

So XI is plus minus one so we can really think of this as the following is you have a set of points, like, M points and what you want to do is you're gonna partition them into two groups. Okay. That's one group and then the other group will be this, okay. And we encode that by saying here's where XI is plus one and here's where XI equals minus one. So we're gonna use the variable here, which is XI, which is the plus minus one vector, to basically encode a partition. It's a partition. It's just that it's a numeric data structure to encode a partition, okay. All right. Let's look at what the objective is. The objective is sum XI XJ WIJ and let's just see what it is. So you sum over all pairs. If XI and XJ are in the same partition, what happens, what is XI XJ?

Student: One.

Instructor (Stephen Boyd):One. Okay. And then you add this to this thing. Now, this is something we want to minimize. Okay. Now, if XI and XJ are in opposite partitions, this is negative so I think that means that WIJ is a measure of how much I hates J. Did I do that right? I believe so because if W is very high, it means that if XI and XJ have the same side, you're gonna be assessed a big charge in the cost. I mean, if XI is high and they're in opposite things, you're gonna decrement the cost a lot and happiness is gonna go up. Okay. So WIJ is basically how much I annoys J, but it's symmetric so it's the average of how much I annoys J and J annoys I. Okay. Now, if WI is small, it means they don't care much. So in fact, this now makes perfect sense as you have a group of people, you have social network or something like that and you want to partition it. There would be obvious ones. If the sign pattern in W were such that like everybody liked everybody except one then it would be very simple; you'd just isolate that one nod. But in general, actually finding a solution to this problem is basically extremely hard. You can't do it. So if this was a hundred or a couple of hundred, you can't do it. It just cannot be done. Okay. So that's the partitioning problem and for us, it's gonna be a canonical example of a hard problem. By the way, there's instances of it which are easy, I just mentioned when there's some obvious solution, but I'm talking about the general case here. Okay. By the way, it comes up in tons and tons of other – my interpretation was sort of a joke, but the point is it comes up in tons and tons of real applications, it comes up in partitioning, it comes up in statistics, I mean, just everywhere. So my interpretation was a joke but it's a very real problem with real applications. Okay. Now, the dual function is this. We simply take X transpose WX, we add as the lagrangian tells us to a linear combination of these functions. I write them as XI squared minus one so I get this and I have to calculate – this is the lagrangian and the lagrangian is quadratic. Okay. It's a quadratic function. We're gonna have a very short discussion about how do you minimize a quadratic function. The first thing you do is let's talk about how do you minimize a quadratic form? So what is the minimum of a quadratic form so what is the minimum of a quadratic form?

Student:[Inaudible]

Instructor (Stephen Boyd):What is it?

Student:[Inaudible]

Instructor (Stephen Boyd):It can be negative infinity. I agree. When would it not be?

Student: Within [inaudible]

Instructor (Stephen Boyd):If the quadratic form is positive semi-definite then the only values it takes on are non-negative so it couldn't be minus infinity then so that's exactly the condition. The minimum of a quadratic form is minus infinity if that matrix is not positive semi-definite so if it has one negative eigenvalue, the minimum is minus infinity. Okay. Otherwise, if it's positive semi-definite the minimum is zero because it can't be any lower than zero and it can be zero by plugging in zero. Okay. Let me tell you what the lagrange dual function is for you right now. It is a lower bound on an optimization problem, gives a lower bound. It of course can give you – it's parameterized by lambda and nu by these dual variables. Now, in some cases if you plug in some lambda nu you're gonna get the following lower bound minus infinity. But it's always a lower bound. In some cases, you plug in lambda nu and you're gonna get actually an interesting lower bound that's not obvious. So right now, you should just think of it as a lower bound. Lower bounds can be good, they can be tight, they can be crappy, we're gonna get to all that later. Okay. Okay. I just want to tie together the idea of the [inaudible] function and lagrange duality. So if you have a function with just linear inequality and equality constraints, a problem, and you work out what the dual function is, it's a minimum of F0 plus – I collect this together and multiply by X and then that's, of course, a constant. And what this means is the following. If I focus on this and then go and look up what the conjugate function is, which was the conjugate over X of Y transposed X minus F of X, that's the dual, if you plug in also all the right minuses, you get this. It's equal to that. Now, what that means is the lagrange dual function of this thing is exactly equal to this.

It's equal to that. Now, recall the conjugate functions often have domains that are not everything. It was actually the probability simplex was the domain of it. So that'll automatically impose some inequality constraints in here when that happens, but here's an example. The maximum [inaudible] problem is maximize sum minus XI log XI subject to some inequalities and equalities. And by the way, that's already a really interesting problem because it says lots of things. It says find me the maximum entropy distribution that has these expected values – these are just known expected values. These can be moments, it could be probabilities, it could be anything and these are inequalities on expected values. So it's really quite a sophisticated problem to ask. What's the maximum entropy distribution, for example, on these points that, for example, has the following variance and has the probability in the left tail less than that. You could go on and on and make it a very sophisticated thing. That's the maximum entropy problem. That's this thing. And if you work out what that is when FI of X is the negative entropy here, that's minimized negative entropy, you will actually get the sum of exponentials. So the dual function for a maximum entropy problem is gonna involve a sum of exponentials. Now, if you're in statistics – and I said statistics not probability, this will be very familiar to you because it's a connection between exponential families and maximum entropy and we'll see more of this later. Just a hint. Okay. Now, we get to the dual problem and to write down the dual problem – I mean, it's the dumbest thing ever. If someone walks up to you and says, "I have a lower bound on my problem, but it's parameterized by this vector lambda and this vector nu," and then you say the only interesting thing about lower bound is, "Well, that it's a lower bound," and if someone has multiple lower bounds, obviously the higher the lower bound, the more interesting it is.

So you can just say okay, what's the best lower bound, on the original problem, that you could establish by lagrange duality? What is the best lower bound? We don't know if it's sharp. We're just saying what's the best one and it kind of wouldn't make any sense to really examine any other anyway. All right. That leads you just to this problem right here. Now, I want to point something out. This is always a convex optimization problem no matter what the primal problem was. Oh, by the way, this is called lagrange dual and sometimes it's just shortened to the dual problem here. In fact, people say "the dual problem," the same way we say "the optimal point," even in situations where we don't know that there's an optimal point. We're actually gonna see this multiple dual the way the word is used on the street. There's lots of dual, but we'll get there. For now, it actually really is "the lagrange dual problem." And it says simply maximize the dual function subject to this. The subject to the lambda's being positive, that's all. Okay. Now, often what happens is G is minus infinity for some values of lambda and nu. We've already seen that a couple of times. That is a not interesting lower bound and it's sure not gonna help you maximize something. To find a point where it's minus infinity, you know, this thing could actually be minus infinity, that can happen, but the point is it's not an interesting value. So in fact, often what happens is you pull the implicit constraints in G out and make them explicit. Okay. Now, here for example, let's go back and look at this. The dual function for this LP is this weird thing that looks like this. I drew it somewhere. It was this sick thing here where this thing is kind of going up on a line, but off that line, the thing falls off to minus infinity and we're just simply going to maximize that subject to lambda positive; however, it's easier to simply take the implicit constraint out and you end up with something that looks like this.

Okay. So here's the so called standard form LP and then this is what it looks like when you have actually pulled out this implicit constraint. Technically speaking, this is not the lagrange dual of that. However, people would call this the lagrange dual so you're given a little bit of license to form the lagrange dual and do a little bit of trivial rearrangement and people would still call it the dual or something like that. This is equivalent under a very, very simple equivalence of this. The lagrange dual of this thing is that where G is the sick function. I just want to point this out. Okay. And by the way, let's see what happens here. That is also an LP and let's see what it says. Here I can say something about this problem. If you have a feasible nu here then minus B transpose nu is a lower bound on the optimal value of this problem. This thing says, "Okay, you have a family of lower bounds, please get for me the best lower bound." That's what the meaning of this problem is. This also has beautiful interpretations in a lot of cases. So for example in engineering design, it'll make a lot of sense. X will be a sub optimal design, for example, sorry, any X here, if it's feasible, it satisfies the constraints, but something that's feasible here would be a sub optimal design. Okay.

The nu will have a beautiful interpretation. A dual feasible nu in that case is a certificate on a limit of performance. That's what it is. That's the meaning of nu here. Okay. Actually, we'll see that when you look at a real problem, it will have physical significance. We'll get lots of examples. If this is a question of how bad could something be bad, if, let's say you're at a bank and they want to know, okay, what's the worst thing that could possibly happen, then a lower bound actually gets interesting. Yeah.

Student: Could you please say why that was not, technically, a lagrange dual on the right?

Instructor (Stephen Boyd):This?

Student:

Yes.

Instructor (Stephen Boyd):Sure. That's the lagrange dual, that's why.

Student:I mean, relative to the LP.

Instructor (Stephen Boyd):No, they're not the same thing. They're not the same thing. This is minimizing a function, which is a weird function. It's equal to minus B transpose nu provided that A transpose nu plus C minus lambda equals zero, okay, something like that and it's minus infinity otherwise. Okay? So that's what this is. And by the way, if you were very careful, it would make a different. Let me explain that. For example: suppose I throw in a nu for which A transpose nu plus C is not a non-negative vector, okay? Then in this problem when I say how about this nu, how do you like that, what is sent back to me is actually the infeasible token is sent back to me saying your nu is infeasible. Okay. Over here, it's actually more interesting. Over here, if I throw such a nu in or whatever, what comes back to me is the object function sends me an OOD token, Out of Domain. Now, that's a concave function and that means it's minus infinity. You get two slightly exceptions are thrown in this thing. But I want to point out that these are just – you can call this just silly semantics and all that if you like, but it's very important to understand these are not the same problem. By the way, don't focus on these minor things. That's something you can think about, you can read this, think about it on your own. Don't let silly little technical things get in the way of what the picture is. The big picture is you have an optimization problem and you form another one called the lagrange dual. That lagrange dual problem, essentially, is saying what is the best lower bound on the optimal value of the first one I can get using the lagrange dual function. That is what's important. Okay. So now we get to the idea of weak and strong duality. Now, weak duality says that "D" star is less than "P" star. Now, here, let me see how this works. Okay. So in this context, the original problem is called the primal problem and the lagrange dual is then called the dual problem. Okay. So that's the primal and the dual and we'll call – well, we've already assigned the symbol P star to mean the optimal value here. We're gonna let D star be the optimal value of the dual problem. Okay. So optimal value of this is gonna be denoted D star. You always have D star as less than P star.

Why? Any dual feasible point is a lower bound P star so the best one is also a lower bound. This is called weak duality. It's called weak duality because let me review the deep mathematics required in establishing this, right, it hinged on properties such as the product of two positive numbers is positive in the sum of positive numbers.

So it's weak because you can explain it to somebody in junior high school. I mean, they might not have taken those 14 steps, but the point is it has nothing in them that's hard so it's called weak. Okay. All right. That's weak duality. All right. It always holds. Convex, non-convex. It's absolutely universal. It could be stupid. You could, indeed, have D star equals minus infinity in which case your best lower bound is of no interest what so ever. Okay. That can happen, but this is always true. Okay. Now, if we go to the partitioning problem and we ask what is the best lower bound on the two-way partitioning problem you can get from the lagrange dual you will form this problem. That is a semi-definite program. And now, things are interesting because, although this is something that was not known 15 years ago, and absolutely inconceivable 20 years ago, I can tell you this, this SDP, you can solve it, people can solve it. You can solve it like that for a thousand variables. No problem here. And if you knew what you were doing you could go, easily, to problems with 10,000 and 100,000. The point is, you can solve this SDP and you will get a lower bound on the two-way partitioning problem. That is fantastically useful if you couple that with a uristic for partitioning. So you do some crazy uristic, there's lots of uristic; some of them work really well by the way. Now, you don't expect it to work all the time because you are solving, after all, an NP hard problem in general, so you don't expect it to work well all the time, but what happens is you'll do a partition and you'll say, "Here's my partition and here's the number I got," whatever it is. It's the X transpose WX and you want to know could there be a better one. You can solve this SDP and in fact, you'll see in a lot of times the numbers are pretty close. Okay. At least it's a good thing to know. You would know I have a partition, but there's no partition that's more than – I'm at most such and such sub optimal.

And you might just say, okay, that's good enough. All right. Okay. Strong duality, this will not rely on junior high math, okay. Strong duality is going to be that that lower bound is tight. That says, there's a lower bound that goes all the way up to the optimal value. That's strong duality and we'll see what its equivalent to, but that is not trivial. And by the way, it often doesn't happen. Okay. So in two-way partitioning problems, by the way, if it were true there, you'd have P = NP because this problem we can solve in polynomial time and so in fact, if P star were equal D star – and in fact, there's even approximation. If you know about complexity and you have something that's not even approximable or something like that then that tells you that you can't even get something where you can bound the gap or something like that but I won't go into that. Now, here's an interesting part. When a problem is convex, you usually have strong duality. Okay. So that's actually amazing. That's gonna actually have a lot of implications. It's gonna be equivalent to, by the way, it's gonna involve the separating hyper plane something. We'll see what it connects to. There are multiple books, multiple courses, not here, but at some other schools; you can take entire courses, read books, thousands of papers that elaborates on this one word, usually. Okay. Now, basically these are called constraint qualifications. So a constraint qualification theorem goes like this. It says if the primal problem is convex and then you insert your constraint qualification here, okay, then P star equals D star. That's a constraint qualification. You could devote your life to this. On occasion, these issues actually do come up, but maybe less frequently in applications than the people who devote their lives to it would like to think. I'm saying that of course because their grad students will watch this and then alert them to it.

So I'm just making trouble. Now, by the way, if you're in this industry, sub industry of constraint qualifications, then this like the big, the sledge hammer, the most unsophisticated one there could be possibly be, this is the basic one that everybody knows. Okay, this is the least squares or something like that of the constraint qualification worlds, its Slater's Constraint Qualification, although, actually, the correct name here would probably be Russian, but we won't get into that. So let's call it Slater's Constraint Qualification and it says this, if you have a convex problem like this, it says if there is a strictly feasible point, if there exists one, then P star equals D star. Strictly feasible means not just that you met the inequality constraints, but you do so with positive margin for each one. That's the condition. Okay. Now, I should add that basically, it's completely clear, that for most problems that covers everything in engineering, pretty much, I mean, as much as people would make fun of Slater's Constraint Qualification and give you reasons and they could make examples up why it's not sophisticated enough and sure enough, there are problems where you don't have a strictly feasible point, but for most problems that come up in engineering, anything in machine learning, pretty much anything, this makes perfect sense, right.

For example, if the third inequality was a limit on power, it doesn't make any sense to say – just think about it, right? If Slater's condition failed to hold, it means their existing circuit dissipates 100 milli-watts, but there's no circuit that dissipates 99.999999 because if there were, Slater's condition would hold. Everybody see what I'm saying here? If solving that problem relied on these most fantastically subtle facts as to whether strict inequalities held or weakened equalities and one, but not the other held, then I got news for you, you're not doing engineering, you're not doing statistics, you're not doing economics, you're doing something like peer analysis. Okay. So that's my little story on it. Again, there are actually cases where these come up in practice, but they're pretty rare. And mostly, I'm saying this to irritate at other universities, my colleagues, who will be alerted to this, watch this tape and be very angry. But I thought I'd mention this. Okay. All right. So let's go to the inequality form linear program. Here you want to minimize C transpose X subject to X less than B. G of lambda is C transpose X plus lambda transpose AX minus B because I put the B on the left-hand side to make this F less than zero. I do this and I infamize this, but we know how to infamize a affine function. You get minus infinity unless the linear part vanishes so I get this and so this is the dual problem. Notice this is actually not the dual problem. So if there's lawyers present, you would say, "This is a problem that is trivially equivalent to the dual problem," okay, but after a while if there are no lawyers present you'd just say that's the dual problem or something like that. So that's it. Okay. Now, Slater's condition says that if the feasible set – of course the feasible set is a polyhedron and by the way, one possibility is the feasible set could be empty, which in fact, is a polyhedron. What Slater's condition says geometrically is very simple. It says if that polyhedron has non-empty interior, that's what this means, it means, basically, that there's an interior point, if it has a non-empty interior then you have strong duality so you have P star equals D star. Okay. So that's the picture.

Let's look at a quadratic program. Let's minimize X transpose PX subject to AX less than P. That's minimizing quadratic form over a polyhedron, the dual function is this X transpose PX and we're gonna assume P is positive definite. Actually, that's so that I can avoid the horrible way to write down – it's not that big of a deal, but the horrible to infamize a general quadratic function with a linear term because I don't feel like doing it so this will work out. So here the dual function is you infamize over XX transpose PX plus lambda transpose AX minus B here like that and now I minimize over X. Now, the nice part is P is positive definite so I know how to minimize this. It's P inverse times whatever something. I'm not even gonna do it because it's easy to minimize a strictly convex quadratic function so I minimize it. I plug that X back in here and I get this thing, okay, which is I get minus one quarter lambda transpose A, P inverse, A transpose lambda something or other and my dual problem then looks like this. By the way, this really is the dual problem because in this problem, up here, notice that the dual function, the domain is all of our – let's call it RM, it's all of RM. Okay. So in this case, the dual function is domain is everything, which is to say, you get a lower bound for any – if you plug in random numbers lambda and you're not gonna get a trivial lower bound. Okay. You might get a rather stupid one. For example, you might get the lower bound minus seven. Let's talk about the lower bound minus seven here. Why is the lower bound minus seven valid for this problem? Because the objective is always non-negative, but the point is, you get a lower bound and you get this. So that's the dual problem. And by the way, what we're saying here is not obvious at all. What we're doing is we're saying, you want to solve this quadratic program – we haven't yet told you how to do it or how it's done or anything like that, but we'll tell you this, if you come up with any vector lambda that's non-negative and you evaluate this concave quadratic function, you get a lower bound on the optimal value of this thing. This has lots of uses. For example, suppose someone says I know how to solve this problem and you say, "How did you do it," and they go, are you joking, – that's, like, "If I told you, I'd have to kill you." I'm patenting it right now in [inaudible]. Okay. I can't tell you how I did it.

And you say, "Well, why should I believe that that's the optimal X, how do you prove it? You say, "Well, watch this." You say, "Check out this lambda, notice that it's bigger than or equal to zero," and you go, "Yeah," then you evaluate that number and that number is equal to the value up here of the point. That, by the way, ends the discussion. That X is feasible and by the way, you would call that lambda a certificate proving it. Everybody got this? And notice that you didn't have to say how you did it. Everyone got this? And then you'd say, "Hey, how'd you get the lambda," and you go, "Like I'm gonna tell you that either." Now, Slater's condition says the following: If this polyhedron has non-empty interior, then these are absolutely equal then their always exists a certificate proving optimality of the optimal X. Always. So okay. By the way, a very small number of non-convex problems have strong duality. I'm not gonna go into it because it's complicated and so on. This is actually covered in an appendix of the book and I would encourage you to read it. This one is not obvious. And actually, there's a whole string of these. There's, like, 15 of them or something like that and they're just weird things that have to do with specific problems that are non-convex and just happen for deeper reasons to have zero duality gap.

The quadratic ones are the ones we collect at the end of the book in one of the appendices. There are others, you will see them, they're kind of weird and some of them are quite strange. One I've seen recently where it involves complex polynomials of degree four. Right? And then something that should have zero duality gap and it comes down to something in algebraic geometry, but that's always the way these are. These are not simple. This is just to say there are non-convex problems with zero duality gap. A few. Okay. Let's look at the geometric interpretation. All right. So let's see if we can do this right. So we're gonna do a problem with just one constraint so what we're gonna do is we're gonna minimize – I'm gonna write the graph of the problem. What I'm gonna do is for each feasible X or each X in the domain, I'll evaluate this pair. So although the problem may be happening in a hundred dimensions, for every X, I'm gonna plot a point which is in this plane; and one, basically, this tells you the objective value and this tells you the constraint function. So, basically, everything over here corresponds to feasible. Okay. And then the height corresponds to the objective value, so quite obviously, that's the optimal value. Any point that ends up being colored there is optimal. Okay. So that's the optimal value, P star. Everybody see that. So that's the idea. So that point really has a very nice objective value, but it's infeasible because it's constraint function is positive. Okay. So that's P star. Now let's see what the dual is. How do you get lagrange duality in this picture? Well, lagrange duality works like this. You minimize F0 plus lambda F1. Now, on this plane, that corresponds to taking something here like this an it's got a slop of – is it one over lambda or something like this, let's see, it's slope minus lambda so I take something like that.

So for example, if you fix lambda and then ask me to evaluate the dual, what you do is this. You fix a slope here and you march down this way until you just barely leave this set, and that would be right there. Okay. And then when you work out what G of lambda is, it's this intersection here. Okay. So this is G of lambda and now the dual problem says, "Optimize over all lambda," so if lambda is zero, you get this. You go down there and G of zero is this number right here, which is indeed a lower bound on P star, it has to be. Okay. Now, I crank up the slope and as I crank up the slope G is rising and it keeps rising until you just hit here, this point, at which point here its right there. Okay. Now as I keep increasing lambda what happens is the optimal point is actually here and this thing is rotating around – it's not a fixed point, it's rolling the context, but because it's got sharp curves, it's just rolling just slightly. It's rolling along here and as I increase lambda, G gets worse and worse. In fact, if lambda is huge, it looks like this and G is very negative. It's still a lower bound, just a crappy one. Everybody see this. So D star is that point. Questions?

Student:[Inaudible]

Instructor (Stephen Boyd):We're gonna talk about that, but it depends very much – so for example, in a non-convex primal in two way general partitioning problems, NP is hard, but the dual is a SDP. That's easy. In that case, it can be infinitely far away. Now, in the case of a convex problem, now it gets interesting. So in a convex problem, you will see later that they both solve the problem and a lot of people get all excited and they go, "Oh, how cool, I can solve my problem by the dual." It turns out that if you really know what you're doing, the complexity of the primal and dual are equal if you really know what you're doing. You will in about four weeks. Three. Whatever it is. Yes?

Student: How did you rule out the bottom point for P star? You can't just say it was that.

Instructor (Stephen Boyd):How did I do it?

Student: Yes.

Instructor (Stephen Boyd):Well, the first thing I asked is I asked – this shows you the objective and the constraint function for every possible point in the domain, okay, now, that points not good, for one thing, it's got a high objective, but it's also infeasible. Anybody who landed on the right here is infeasible. So in fact, these are very interesting, but they're not relevant as far as the optimization problem is concerned so we simply look at these. Now, every point that got shaded in here is feasible. Okay. The height tells you the objective value and so you want the lowest point among these. That's clearly right there and you go across here and that's P star.

Student: Why would the right-hand be infeasible?

Instructor (Stephen Boyd):Because your first coordinate here is your constraint function and F1 has to be less than or equal to zero. That's what it means to be feasible. Okay. So that's the picture. So here you have a gap. By the way, this thing strongly suggests something very interesting and you can see why convexity of the problem is gonna come in. When F1 and F0 are convex, this weird set G – now, what I'm about to say is actually not true, but it's close to true – that weird set G is convex, okay, when something is convex, you have a gap here because this blob is non-convex so if this thing had to be convex you can't have a gap. Everybody see this? That's what is gonna happen. Now, I'll tell you the truth. G is actually not convex, but its lower left corner, which is what we care about, is. Now I've corrected it and said the truth. By the way, you can also see how Slater's condition works so if you take not G, but A, that's the set of points that you can meter beat in a bi-criterion problem, so basically, if you take A then color in all these points here and now you can see A will actually be convex if that's convex and that's convex so A will have to look like this. Slater condition says that somewhere A goes a positive amount or it goes into the left side. These are the kinds of things you would study in a one of these whole courses on this topic. So that's the idea. So you can even get how Slater's condition connects to all of this. Okay. I'm gonna mention one more thing. We'll get to one more topic. It's a complimentary slackness. So let's assume that strong duality holds and actually, I don't care if the problem is primal or feasible. Okay. Convex. What I said made no sense whatsoever so let's start over. What I meant to say was I don't care if the primal problem is convex. That's what I meant to say, but it just came out a weird permutation. Okay. So I don't care if the primal problem is convex, of course the dual problem is always convex. So let's assume strong duality holds and let's suppose X star's primal optimal and lambda star and nu star are dual optimal. That says this. By the way, this is basically what it comes down to, it says X star is an optimal point, lambda star and nu star, you can think of then as a certificate establishing optimality of X star. Okay. By the way, these ideas, we're gonna use them from now on. They're gonna come up computationally. All algorithms are gonna work this way. All modern methods – you haven't done it yet, but whenever you solve a problem, it doesn't just say here's X and you have to trust the software or whatever.

It doesn't work that way, although you haven't seen it return you yet. They also return, no exceptions, a certificate proving that it's the solution so you don't have to trust the implementation. Everybody see what I'm saying? These ideas are gonna diffuse through everything we do. So basically you think of that as an optimal point, optimal design, whatever you want to call it, this is a certificate proving that's optimal because that's what it is. That's a lower bound on P star, that's a point that's feasible and satisfies and has objective value equal to this lower bound of P star, therefore, it is P star. Now, by definition this thing is the infinium over all X of G with these optimal lagrange multipliers. Okay. But if it's the infinium over all X, it's certainly less than or equal to this when I plug in a particular X and I'm gonna choose to plug in X star. Okay. So I plug in X star and I have the following. Very interesting. This says F0 of X star is less than or equal to F0 of X star plus something where every term in that is less than or equal to zero. Okay. And every term in that is zero. So this one is not relevant. Okay. We'll get to that. Okay. Yes, everything here is zero. And now you say, wait a minute here, if this thing is less than or equal to that thing and that's the same as that, then they're all three equal and we have no choice but to conclude that the sum of lambda I star times FI of X star is zero. Okay. But there's more than that. Wait a minute. This is a sum of numbers, all of which is less than or equal to zero. If you have a sum of numbers, which are less than or equal to zero and it's equal to zero, there's only one conclusion; every single one of those numbers has to be zero. And that says the following: lambda I star times FI of X star is actually equal to zero for all of these. Okay. And that's known as complimentary slackness and what this means is the following: it says if you have any primal optimal point and any dual optimal point, the following must hold; if the optimal lagrange multiplier is positive or zero then that thing has to be tight. If a constraint is loose at the optimal point, these lagrange multipliers have to be zero. Okay. So this is gonna have lots of implications and when we give other interpretations of what all this means, it's all gonna tie in, like, with these things being prices for example. But we'll quit here for today.

Student:[Inaudible]

Instructor (Stephen Boyd):Exactly.

[End of Audio]

Duration: 77 minutes

ConvexOptimizationI-Lecture09

Instructor (Stephen Boyd):Okay. We'll continue on duality today. One thing though before we get started, I've received from a variety of sources, feedback that the homework is maybe just a little bit too much? I don't believe – I think it's fine, frankly, but – so the question would be – how many people would support, like, suppose we were to hold back. By the way, there's a limit to how much we can hold back. It's gonna be very mild. But suppose, I mean, the teaching staff, we could exercise a small amount of restraint in assigning the homework. Would there be general enthusiasm for this, or? Yes. Okay.

So let me limit – there's limits to our effectiveness. So we'll – but we'll see what we can do. What's the lower bound? Oh, no. No it's not very far. We have a very small gap between what's feasible and what the lower bound is. We might back off, like, one or two – one problem or two. Yeah. You know. Sorry. So that's what we'll – that's what we'll do. I mean the fact is this is how you actually learn the material, so don't see any reason to back off more than that, so okay.

Well, since I didn't hear, like, people screaming and yelling, that's sort of consistent with our information gathering, that it's – that some people think it's a bit too much. Actually, I got some complaints from faculty members in other departments who said that all research in their department had come more or less to a standstill because of this class, so. Interestingly, they did not advocate backing way off the homework, interestingly. They said back off a little bit. That's all. That was their suggestion.

Not that I'm under any obligation to do something a colleague in another department says. Okay. All right. Let's – you can go down. We'll talk about complimentary slackness. By the way, these – the topics we're covering right now involving duality – full optimality conditions, things like that, these are very tricky. So you actually have to go over these a couple of times to get the full exact logic of what implies what.

It is quite tricky. So just – just to let you know, actually I read this stuff very carefully a couple of times actually just so I can remember sort of exactly what implies what because it's kind of subtle. Okay. And let's see if I can end up not confusing myself today. Okay, so assume strong duality holds. Now what that means is we have a primal optimal point, X star, and we have lambda star, nu star, which are dual optimal. [Inaudible] these are just any primal optimal, any dual optimal point.

And of course what it means for strong duality, this means that the value achieved by X star, which is feasible, is equal to G of lambda star, nu star. Now that's a lower bound on the optimal value. So if you've got a point that's feasible and has an objective value equal to the lower bound, you're done. It's optimal. So you can think of lambda star, nu star as a certificate proving X star is optimal. By the way, the same is true the other way around.

X star is a primal certificate proving lambda star and nu star are optimal. Okay, so let's see what this means. G star of lambda star, nu star is by definition – it's the infimum over of this thing. I know we went over this last time, but I just want to sort of go over this again quickly. This is the infimum over X – over all X here of this lagrangian. Now, of course, that has to be less than or equal to the value of the lagrangian if you plug in X star.

You plug that in and you get this. But X star is feasible. But therefore, F I of X star is less than or equal to zero. The lambda I stars are bigger than or equal to zero. They have to be. That's dual feasibility. So this whole term is less than or equal to zero. This term is equal to zero because these H I's are zero. Therefore, this thing here is less than or equal to that because it's this thing here plus zero, plus something less than or equal to zero.

So it's less than that. Therefore, everything in between this chain has to be equal – every expression because if something is blah, blah, blah, less than this, less than that, back to something again, everything in between has to be equal, and that tell us, since that's zero, it says this thing's gotta be zero. Now that's a sum of a product of non-negative numbers with non-positive numbers. And that zero – that happens only if every single product is zero, okay?

Let me just – today this'll make sense. Last time it was just an algebraic fact. Today there'll be lots of interpretations under which this makes sense. What that says is the following – for each eye – for each constraint, either – lambda I star is zero, so either the optimal lag range multiplier is zero, or the constraint is tight. That's what F I of X star equals zero means. So you either type – or the constraint is zero.

And you can – logically you can twist the logic around and say it many other ways. You can say things like this, you know, if lambda I star is positive, then the constraint is tight. If a constraint is slack, that means F I of X is less than zero, then you must have lambda I star equals zero. And later today we're gonna have some interpretations of lambda I star that will make this completely clear and totally intuitive.

So this is called complimentary slackness, and what it really means is that the vector of these Fs and the vector of the lambdas actually have complimentary sparsity patterns. So a sparsity pattern in a vector tells you what entries are zero and what are non zero. And this has says they're complimentary. They cannot have non zeroes in the same entry, so that's the picture.

Okay. This brings us to something called the KKT conditions. They're some fun history here. Kuhn and Tucker are sort of well known – who had worked these out at some point. I mean these are kind of silly because I think, for sure, there were people in Moscow for example who know these earlier, but it doesn't matter. And then actually somebody uncovered a 1938 master's thesis. This poor guy Karush – that had everything in it. And actually, miraculously, they added his name to this.

So these are now – in the west it's called the KKT conditions. So this is basically a generalization of the very simple fact that for an unconstrained problem, the necessary and sufficient conditions for optimality, or a convex function, is if the gradient is zero.

That’s it, period. You already know that. Now the question is what about with inequality restraints and so on. And these are for differentiable – differentiable F I and H I.

Okay. Here it is. So here it is. This is – you have to have the following. Obviously, you have to have primal feasibility. So X could not possibly – by definition it could not be optimal if it weren’t feasible. So feasible means that it satisfies the inequality constraints and the equality constraints. That’s feasible. Number one. Dual constraint – dual feasibility says that the lambda Is are bigger than or equal to zero. Complementary slackness says lambda I F I is zero, and then the final one is gradient of the lagrangian with respect to X vanishes. That’s this expression here.

So this is simply – you might write this something like this, you know, partial L, partial X, something like that. That’s a gradient. I guess it would be all right this way. Gradient X L equals zero, okay? So these four conditions together are called the KKT conditions. And, in fact, what we just showed is the following: if you have a problem – by the way, convex or not, either way, and strong duality holds, then the KKT conditions will hold for any pair of a primal and a dual feasible – sorry – a primal optimal and a dual optimal – here, got a question? Oh, okay. Sorry. Okay.

So that’s the – so if strong duality holds, the KKT conditions must hold, okay? Now, for convex problems, it’s necessary – it’s sort of necessary and sufficient, and you have to get sort of the logic exactly right. So let’s see what that means. Now suppose you have a convex problem and some points, X tilde, lambda tilde, and nu tilde satisfy KKT, okay? I’m writing them as tildes because if I put a star on them it sort of presupposes by the notation. It implies that they are optimal.

They will turn out to be optimal, but for right now I’m gonna use tildes, which are more neutral, okay? So suppose – now X tilde, lambda tilde, nu tilde satisfy the KKT conditions. Then we’re gonna argue that, in fact, they’re optimal, okay? X is primal optimal, lambda nu is dual optimal. Okay, so the way that works is this – from complimentary slackness – if complimentary slackness holds – if all of these conditions hold then, in fact, lambda I F I is zero, right, because for each I, either lambda I is zero or F I is zero, okay?

These are – all – the other things, the nu I H I – those are all zero, and therefore, L of X tilde, lambda tilde, nu tilde, is equal to F zero of X tilde, okay? So we’ll start with that. Now the fourth condition, that’s this gradient condition here. If this holds, then look at this. F zero plus lambda – sum lambda I F I, plus some nu I H I is gonna be a convex function. Why? Because the H Is are affine. You can multiply them by any number and still have an affine function. The F Is are convex. I can multiply them by any non-negative numbers, still get a convex function. I can add up a bunch of convex functions to get a convex function differentiable.

This condition says that the convex function F zero of X plus sum lambda I F I plus sum nu I H I, that this thing has a gradient that vanishes. If that has a gradient that vanishes, it’s a convex differentiable function, then at that point, that point is the minimal.

Therefore, that says that X tilde here minimizes – is a minimizer of the lagrangian, and therefore, if I evaluate L of X tilde lambda tilde nu tilde, I get G. Hey, I'm done because now I've just shown that these two are equal. That's an objective – that's obviously upper bound. On the optimal value, that's a lower bound. They're equal. It's optimal, okay? Now there's also the question how Slater's condition comes in, so let's review what we've seen. We've seen the following. If for any problem, convex or not, that you have a strong duality of [inaudible] – If that's the case, then the KKT conditions hold. And the converse is false. Actually, in other courses, not this one, you'd study that like crazy, and you'd have numerical methods that would search for KKT points, and there'd be false KKT, you know, KKT points that aren't optimal, and so on. But as far as we're concerned, any problem you have strong duality, KKT conditions hold. That's number one. Number two, if the KKT conditions hold for a convex problem, you have – then those points are a primal dual optimal pair, period. Okay? And the last one says this. You have to have Slater's condition. Slater's condition says that if Slater's condition is satisfied – that means there's a strictly feasible point – there exists a strictly feasible point, then it says the following. It says that a point is optimal of, and only if, there exists lambda nu that satisfy KKT conditions. So that's the full logic of it. Question?

Student:

When you say [inaudible], do you mean [inaudible]?

Instructor (Stephen Boyd):No, no, no. What I'm saying is – just let's go very – so here, the KKT conditions – I mean you have to be very careful about how you state it to say it's necessary and sufficient, so I'll state one carefully. Ready? It says this. If a problem is convex – in fact, I'll just say the true statements. Ready? So the trust statements are as follows. If [inaudible] assumption on convexity. If X – if a problem has strong duality obtains for it, then the primal and dual optimal points will satisfy KKT conditions, period.

By the way, this is very much like saying when you minimize a non-convex differential function, at the minimum, the gradient will vanish. That's true. The converse is false, of course. Same here. Okay, so that's the analog of that classical calculus statement. Then the next one says this. If the problem is convex, and a set of points satisfy the KKT conditions, then in fact, they are primal and dual optimal. So at that point, if you wanted to, you could say something like this.

For a convex problem, the primal dual optimality is absolute if, and only if, KKT holds. So that's one way to say it. Now if you wanted to have a one-sided statement to say what's the condition – I mean, you know, if you think about it, all of the duality stuff we've constructed is sort of – well, we've constructed it. I mean after a while it will become quite natural, but it's sort of like the [inaudible] transform. It's not – it's a cognitive construct. We made it up.

So after you start thinking in the [inaudible] transform, then someone is allowed to give you an answer that involves the [inaudible] transform. Once you start thinking about duality, which you will, I promise, in the next couple of weeks or whatever, then it'll be natural for me to just say the statement I said. However, if you just say, look, I don't care about – what's this duality? I don't even want to hear about it. I have my optimization problem and I want to know when is X optimal.

Okay? So if you wanted a one-sided statement that only focuses on X, then it has to be this one. It would be something like this. X is optimal if, and only if, there exists lambda nu that satisfy KKT conditions. That is almost necessary and sufficient conditions for a convex problem. It's not quite because you have the possibility that, for example, Slater's condition doesn't hold, and you have positive duality gap and all that.

Student: Where does convexity enter in?

Instructor (Stephen Boyd):Do what? What I just said?

Student: Yes.

Instructor (Stephen Boyd):When you remove the convexity assumption, all statements go away except the necessary ones. Right? So it's exactly like sort of going back to calculus or whatever and gradient vanishing. So it's got the same status, works the same way, everything like that, so.

Student: But I mean how did you get that [inaudible] condition?

Instructor (Stephen Boyd):Where'd I get what?

Student: Where did you get this statement?

Instructor (Stephen Boyd):Which one?

Student: It's in parentheses, the one [inaudible].

Instructor (Stephen Boyd):Here?

Student: Yes.

Instructor (Stephen Boyd):Oh yeah, I'm just saying, look, the fourth condition is this, provided it's a convex problem means F zero's convex, F I's convex, H I's affine. Also, lambda I are positive. Therefore, this function here is convex. Gradient of a convex function vanishes, differentiable, done. It's minimized. So that's the logic there. No – I mean there's a lot of – I mean it's not totally straightforward – all the logic here. I believe I have now confused everyone. I don't see anyone who's not confused. So that's a good time to move on.

Okay. So there are some examples of this. Let's do a quick example of an application of this. Is water filling – this comes from communications and information theory, but actually there's problems like this all over the place. In fact – well it doesn't matter. But we'll just look at it as a canonical example. And let me just explain what it is. You're maximizing the sum of the log of X I plus alpha I. And I can even sort of – I can give – I mean I won't do a good enough job to explain to people who don't know about this, but for those who do, I'll connect it.

Log of X I plus alpha I, here you're doing something like you're allocating power to a bunch of transmitters. By the way, if you don't care about communications, ignore what I'm about to say. You're allocating a total amount of power, say one watt to a handful of transmitters, or across different channels or something like that in the communication system. Log of a constant plus a signal to noise ratio is actually gonna be your – is proportional to your actual bit rate obtained on that channel.

So this says maximize the total bit rate by allocating some power across some channels. Okay? By the way, this is used, for example, now in DSL. It's used, actually, everywhere, okay? And I'm not talking about used by professors, I'm talking about, like, it's used when you use DSL for example. So – okay, so otherwise, for everyone else, it's just, you know, it's just maximize the sum of the logs of X I plus alpha I. Subject to this.

And you can even get some intuition about how this is gonna work. Here, you know, obviously log is concave, so you have sort of diminishing returns that if you look at how much power you put in, or, you know, how much X I you allocate here, if alpha I is small, then putting some X I into that channel is gonna give you some – a very nice rate. Once alpha I is big, the incremental return is gonna be very small, right? So it's kind of clear that you want to put the – you want to allocate X because we're allocating a total of one here.

You want to allocate a bunch of the Xs to the small alpha Is. I mean that's clear. Exactly how much – how to do this is not clear. But we'll see how that works. Well, if you write the KKT conditions this is very, very simple. It's this. It's that X of course has to be primal feasible. That means it's a probability distribution. It adds up to one, and it's non-negative. There is a – we have a lag range multiplier lambda associated with this – these inequalities. And we have a lag range multiplier nu associated with this equality constraint. And you just write out what it is.

I mean you write this thing plus sum – actually minus lambda transpose X. That's the term because this term, when you want to put it into lagrangian, becomes minus to flip it around to make it negative. It's minus X is less than or equal to zero. So this is minus lambda transpose X plus nu times – nu is a scaler – times one, transpose X minus one, okay? Now you differentiate that with respect X is separable in X. Right? The terms – it's completely separable in X. So partial L, partial X I is zero, is this. It's nothing but that.

Okay? Then you have the complimentary slackness. It says lambda is – lambda I is zero if X I is positive, okay? Assume – by the way, we'll get later this lecture to what lambda I is, and you'll have a beautiful interpretation as in communications as well. It'll work out beautifully here as to what lambda I is. Okay. Now let's – so this is the KKT conditions.

They have to work here. By the way, the stronger form of KKT conditions says this. Linear inequalities sort of get a free pass in the strong form of KKT conditions.

So weak form of KKT conditions says there must be a point that satisfies the inequalities strictly. Now of course there is here. Obviously there is. You just take X equals one over – X I equals one over N. So the strong form of KKT holds here. But you don't even need that because if the inequalities are linear, they sort of get a free pass in Slater. Okay? They go back to just mere feasibility. So – okay. So if that Slater holds so everything's gonna work. Strong duality obtains everything. Okay, so the conditions are this. Lambda's bigger than and equal to zero, complimentary slackness, and this is this gradient condition. But let's take a look at this.

That's non-negative, okay? So let's actually try to figure out what happened – if this is non-negative, it says nu for sure is bigger than one over X I plus alpha I, period. Okay? Now one over X I – sorry – nu is less than or equal to because you add a non-negative thing to get nu. So nu is less than or equal to one over X I plus alpha I, but whenever X equals alpha I of X I goes between zero and one, it varies between, you know, one over alpha, and one over alpha plus one.

So, in particular, if nu I were less – if nu – sorry, if nu is less than one over alpha I, then here you absolutely have to – you must have lambda I equals zero, period. And in effect you get X I. X I is one over nu minus alpha I – just by solving these equations. I'm just arguing from the KKT conditions, okay? Now if nu is bigger than one over alpha, then lambda I has to be equal to this, period, and you have to have X I is zero. Actually, the argument really goes like this.

If [inaudible] you have to have X I equals zero, and therefore lambda I is this. So, okay, but now you have an explicit formula for X as a function of nu. Nu is a single scaler variable, and you basically have this. You have – dual – primal feasibility says ones transpose X is the sum of this because you get a formula for this – for what X I is. It's just max of zero and one over nu minus alpha I. That sums up to one. This function here is monotone increasing in nu.

And therefore it has – there's a point where this is equal to one. So, by the way, this is sort of a classic example – this is what you'd find kind of in a, you know, to me this is kind of old style calculus or something like that. But I give it as an example. Let me tell you why I call it old style calculus. It's one of – what is calculus? Calculus goes like this. You say, look, here's what a derivative is, here's what a lag range multiplier is, here are these – you write down all these conditions. It's like differential equations or something, and it works like this.

Then in that class, they show you 12 of the 17 total cases known to mankind where you can actually solve those equations. And you know what I'm talking about here? Right? So you have a differential equation, and you look at 12 of the 17 that you can actually solve. No one who actually does anything solves things analytically. No one has done that since maybe the 1940s or something like that. I mean some fantasy from the 18th century, or something like that that persists, okay?

So a lot of people – they show this as, you know, they say, look, isn't this – so you can do it analytically. Actually, this is all quite silly because, for example, I just add one little minor variation on this, and in fact, you can't do it analytically anymore. But the computational complexity of solving it is insane. So as I just mentioned, this is kind of a classical one where you can actually write down the KKT addition and solve it. That's really not the point of the KKT conditions. I just wanted to emphasis that.

We'll see later in the class how, you know, people – it's actually even funnier than you think because people say things like no – defending so called analytical solution they'll say things like, oh no, no, but an analytical is much faster. Actually, that's false in fact. That is just completely false. Actually, computational complexity, if you use – if you solve this problem by methods later in the class, will be faster than, in fact, doing it by the so-called analytic solution.

Student: Do you need to solve the problem for every single alpha I? It seems like [inaudible] –

Instructor (Stephen Boyd):Alpha I are problem data. They're just problem data. So, in fact, they are the only problem data here. So to solve this problem means to be given a set of alpha Is, do some thinking, and to return the optimal allocation X I. So in that sense, yes you do.

Student: Why? Because it has something to do with comparing it to one over alpha?

Instructor (Stephen Boyd):Yeah.

Student:[Inaudible] is just a single number of alpha I since it wouldn't vary a lot.

Instructor (Stephen Boyd):It does.

Student: So which one do you compare it to?

Instructor (Stephen Boyd):Well, in each of these, you're comparing it to one of them, and then you add those up. So that's how that works. The interpretation, which gives us the name water-filling, is this. So what – there's a way to interpret this very nicely. What you do is you make heights alpha I like this, and then you pour in some water here. So you flood this area with, say, a unit of water. Or actually, in general, you flood it in the communication context with – this right hand side is P, which is a total power that you're gonna allocate to some channels.

You flood it, and the height – the depth of the water gives you the optimal X, okay? So that's the picture here. And it's actually kind of nice. It says that basically if alpha is high enough – if you become an island, then you allocate no power. So if you do communications, it makes perfect sense. It says if the signal to noise ratio is really bad, here's the amount of power you allocate to that sub band or whatever – zero. It just doesn't make any sense, okay? If alpha I's really small, you're gonna allocate power to it.

And the amount you allocate goes something like this. Okay. So that's an example. One of the – by the way, handful – very small handfuls of examples where the KKT condition actually leads to a solution of a problem. Right? So, okay. Now we're gonna look at – actually, a very useful interpretation of lag range multipliers, and in fact, it is what makes lag range multipliers extremely useful in practice. And this part is generally not emphasized. Certainly people know it – what is – it's actually shocking how useful this is in a lot of practical context.

So here's what it is. Let's take our original problem – that's this thing here – and what we're gonna do is the following. We're gonna perturb – notice the period. Everything's cool here. See that – min means – it stands for minimize. That's what the period means. Okay, so – [inaudible]. Everything's cool here. So what you do is perturb the problem. And by the way, this is sort of a multi-objective. It's sort of like a multi-objective multi-criterion problem or something like that. But you talk about – in fact, perturbation analysis of a problem is to look at the solution of this problem as a function of nu I – U I and V I.

Actually, that's extremely useful, and actually, generally, should always be done. No exceptions. In fact, we'll see it's – generally speaking, it's locally done for you when you solve the problem whether you like it or not. We'll get to that in a minute. But – so let's actually just stop and say what this perturb problem means. If U I is zero, and V I is zero, this thing is the same as that. Now, if I increase U one – if I make U one equal to point one, for example, then the way you'd say that is you'd say you have relaxed the first constraint because, well, you increased the feasible set. So you've relaxed it. Okay?

And it has lots of interpretations. F one less than or equal to zero might have been a limit on power in a circuit for example. When you relax it – F I is less than point one, it might have said that you just threw another 100 milliwatts at a circuit. Okay? So that's sort of the idea here. It could be a constraint on anything, okay? And you just relaxed it. All right. Now if U I is minus point one, you've restricted – you've tightened the first constraint. So you should think of U Is as loosening and tightening knobs for the problem. Now, of course, if you tighten – if this thing is feasible, and you tighten this, it could be unfeasible. In which case, B star goes to plus infinity the optimal value, okay?

You can't interpret the V Is as loosening and tightening. Actually, what they are is you're shifting. So you're really saying, you know, for example, in a quality constraint – might be an optimal control problem that you hit a certain target. If you mess with the V Is, you're just changing the target point, okay? Is it loosening or tightening? Doesn't make any sense. You're actually shifting the – you're actually shifting the affine constraints in the feasible set. So it doesn't correspond to tightening or loosening either one.

Okay. Now what we do is you look at P star of U V. That is the optimal value of this problem as a function of U I and V I. By the w ay, that's a convex function in U and V. You know that. Okay. And it's really interesting – by the way, it's very interesting to say what this – this is a very interesting thing to look at, and I should mention a few things. If you were just take one U one, and say U one, and plot U one versus P star, you'd get some convex curve, and that of course would be the optimal tradeoff of F one versus F zero. That's exactly what it is.

Okay. Now – so P star, this gives you the optimal value, and you shouldn't confuse what it is. It really means the following. It means, for example, if I were to allocate you another 50 milliwatts of power for your circuit, and – this is the important part – and you were to reoptimize the entire circuit to use that extra power, then this is how much better you would do. Okay? That's very different from a different type – another sensitivity says what happens if you wiggle this but you don't reoptimize?

That's a different kind of sensitivity. This is sensitivity in design. So you – you're given something – or if something – if someone takes something away from you by tightening a constraint, you completely reoptimize and then you evaluate what's happened. And because, in fact, someone could take a lot of a resource away from you, and you could design around it, which is to say, you could reoptimize everything else, even though they've taken a huge amount of resource away from you, you're not doing much worse. Everybody see what I'm saying?

Okay. So – and we're gonna see all of that and how it all works. Okay. So this is the perturb problem. Now the dual of this is in fact just that. You can just work it out. So here it's maximize G of lambda nu. Here it's maximize G of lambda nu minus U [inaudible] actually it's extremely simple to show – minus U [inaudible] minus – because this – what was a zero here becomes a U. I mean it's very simple. Here, you get this? Okay. Now what we're interested in is what can you say about P star of U V? What can you obtain from the unperturbed problem and a dual optimal solution?

And so you get these global sensitivity results. You're gonna do something on this on your current homework. Some of you may already have. But maybe not. You all will have, I think. Actually, I can't remember. Is it this homework? Okay. We're obviously – we're deeply pipelined. We're working on your homework two weeks from now. So the homework you're working on is in our distant, distant and hazy memory. Okay, so you have the following. Assume you have strong duality for the unperturbed problem, right?

And lambda star nu star are dual optimal for the unperturbed problem, then you have the following. P star of U and V is bigger than G of lambda star, nu star, minus this. So you get these – just this affine thing. Okay? And let me just quickly draw a picture just to show you how that looks. Here might be U, as you change U, and here's P star of U, and it basically says that there's an affine function here that lies below C star of U. I mean it kind of makes sense. This is it. Now this condition is really, really interesting. So this star – sorry – P star of zero zero, that's the optimal value of the unperturbed problem. And so it says the following. It says if you tighten or loosen a constraint, or change an equality constraint in a problem, it says it's – actually, this doesn't require convexity here, but it doesn't matter we're gonna be interested in this in the case that it's convex.

Then it says that this optimal value as a function of U and V. You are the tightening parameters near the shifting parameters. It says it's – the change, if I were to subtract that over there, it actually – it's pessimistic. Note that this is an inequality constraint. So it says – what it guarantees you is you will do worse than the following. Worst means you'll be at least this value in optimal value. You will be at least as bad as minus sum U I lambda I star minus sum V I nu I star, period.

It says you will do at least that bad. You might do much worse, including possibly the thing will become infeasible and this becomes plus infinity. Okay? That's infinitely – that's infinitely bad. And now this means – this is a global result. This doesn't hold for U and V small. This holds for all U and V, okay? So okay. Now because it's this global thing, and it's an inequality, basically it's a pessimistic result, it actually is asymmetric. It makes weird asymmetric predictions about what will happen.

It says – basically there's cases where it says the following. Let's look at this. Let's look at a large lag range multiplier. You have a large lag range multiplier, then it says if you tighten that constraint, you are absolutely guaranteed that P star has to go up. Period. It may go up even more. It may go up to plus infinity. That's as high as it can go, meaning you've tightened that constraint and it's become infeasible. But it guarantees sort of in a minimum amount that the performance deteriorates, okay?

What's – interestingly, if you then take lambda I – if lambda I starts large, and you loosen it, you might think, oh, great, then my objective will go way down. And that's actually false. It makes no statement. It might not even go down at all. Okay? This is an inequality. And so you can work out various things here. That's what these are. You just have to look at them and see if you believe them. Now this is a global result. You can also look at this locally.

And you can – let's consider the case where this optimal value of the problem as a function of the perturbation parameters is differential. In that case, you have unbelievably simple formulas, and here they are. And this is it. I mean this is – that's the formula. That's what you want to think about. That's right there. So lambda I star is the partial derivative of the optimal value with respect to U I, period. With a minus sign. Okay?

If it's differentiable – by the way, it is often the case that P star of U V is not differentiable. In which case, the right hand sides here have no meaning. Okay? I mean the global result always holds. Always. This local one requires differentiability. But in terms of interpreting it, it's just beautiful. And let's actually just talk about what it means. Actually, let's look at complimentary slackness. Ready? Let's do this. Let's solve an optimization problem, and let's suppose that F three of X star is equal to minus point one. Okay? What does it mean?

It means that the third inequality constraint is slack. It's not tight. Okay? Now let me ask you this. Suppose I go back to my original problem, which is this, and suppose I perturb the zero. Matter of fact, let me tighten it, or let me loosen it. Let's loosen it to plus point zero one. How does P star change? Not at all because the same X star is optimal. So P star doesn't change at all.

Can I – I can tighten it. By the way, I can tighten it by, for example, point zero one. I can replace zero with minus point zero one. If I replace zero with minus point two, what can you say? Actually, we can say very little, other than it will get worse or stay the same. Okay? So it could actually stay the same, and everything in between is possible. Okay. What is lambda three star?

Lambda three star is zero. It has to be zero. And you can – well, first of all, you could just say, look, complimentary slackness. Look, minus point one times lambda three star has to be zero. That's complimentary slackness. Obviously, lambda three star is zero. Okay? But in fact, you get to the sensitivity. Lambda three star equals zero means you can wiggle the right hand side of that constraint and it will have no affect whatsoever on the optimal value. Everybody understand that?

And it's completely clear. Okay? So this makes – so now you know what it means if you have a big lag range multiplier, optimal lag range multiplier, or small. Big means that constraint – first of all, it better be tight. Has to be tight, okay? It better – it's tight, and it says that constraint is – well, let's see. So far, we've had a qualitative idea of tightness of a constraint. You simply solve the problem, and you look at a constraint – an inequality constraint. If it's equal to zero, we say it's tight.

If it's less than zero, we say it's slack, or not tight, okay? You now have a numerical – a quantitative measure of how tight it is. Right? This is fantastically useful. Let me explain sort of in practice how this works. You know, if you start doing this, what'll happen is you'll have a whole bunch of constraints. And you'll solve a problem, and we'll look at – let's just look at – you look at F one of X star. You look at F two of X star, and so on. And you'll look at F 10 of X star, okay?

And these might be – let's put down some things these might be. Well, let's suppose that all of these are actually zero, okay? What does it mean? This means that when – at least, for the solution you found, X star, all of these constraints are tight. You know, that's your constraint on power. That's area. You know, I don't know what this is. Just some timing – they're all tight. Okay? Now by itself, this is like – I mean it doesn't tell you that much. If I write down the optimal lag range multiplier, things are gonna get a lot clearer.

So suppose I write this, right, and this is 10. Oh, this doesn't matter. Let's make this three. There you go. Right? And this one is point zero two. Now what does this mean? We know that the first constraint, second, and third, are tight. Okay? And if you solve these numbers – and I don't want a precise mathematical statement, I want just what would this mean in the context of solving that problem. It has a very strong meaning. What does it mean? And I want just a rough statement. What would you say about this?

Student:[Inaudible]

Instructor (Stephen Boyd):Right. It says that the set – you would say something like this. Someone would say, well are all the constraints tight? You'd say yep. But the second one – I – this sounds weird – is much tighter than the other two. I know that sounds weird. It's a violation of, you know, it'd be like saying is that matrix singular. Right? And then if you can – if you're – in certain context you're allowed to say things like almost, or nearly, or effectively. Okay? And that means you're talking about singular values, or a small – or a very large condition number. And it makes a lot of sense. You can say the same thing here. Okay?

And what you would see here is that, for example, F two is the constraint. If someone were gonna give you some more resources, and you wanted to ask for more resources, you'd actually ask for F two – whatever F two represents, that's the resource you'd ask for. If someone said we're tightening budgets, take something away, then you'd say, okay, please let's make it resource one, for example. By the way, doesn't guarantee anything. Okay? Nothing is guaranteed. What this says is that this right hand side's zero in the inequality can be wiggled a little bit, and at least locally, I can wiggle a little bit and have essentially no change in the optimal value. By the way, that does not mean X star. You have to do a complete redesign. So in fact, the correct interpretation is you can wiggle U one a little bit. And the redesign will work around it. The concept of the redesign is actually extremely important because that's what this kind of sensitivity is. So basically, someone takes away a little power from you and you're like, okay, no problem. And you redesign the whole thing to basically achieve all of the other specifications. It's very little worse, but it now takes up less power or something like that. So that's the idea. And so this is just, I mean this has huge implications in practice, and in a lot of cases it's just not – well, it's a disconnect, right? Anyone who knows anything about optimization knows about these, and I – almost universally, people who do optimization in application context don't know about these. So there's – it's like a perfect disconnect here. Okay, so now you have – this is sort of – this is the correct interpretation of lag range multipliers, okay? And in fact, if you go back, you get beautiful, beautiful interpretations, for example, here in water-filling. The lambda I now have a beautiful meaning. Lambda I is grade. It's basically – you're violating this constraint. That's the same, by the way, of sort of reducing alpha I. Sort of basically says, you know, if I were allowed here to sort of reduce my – that's something like a noise level, or something like that. If I were to reduce my noise level, how much more net total bandwidth would I get for the same power? And that's exactly what lambda I is. Gives you the partial derivative of that. All right? So it's literally in, you know, bits per second per watt is the unit of lambda I here. Okay. All right, so this finishes this – other interpretations of duality – they're in the book. You should actually read this. I mean just read them because they're just supposed to give you a feeling for what it is. It's very important you have a good. I mean, in addition to knowing all the math, have a good feeling for what these mean. They're – also they generally have interpretations of things like prices. And these examples are given. There's also examples in mechanics and other things. You need to look at these. We'll see lots and lots of examples of these. It'll come up in all sorts of context.

Student:

[Inaudible].

Instructor (Stephen Boyd):

Yep.

Student: From lambda two, or from F two, did you mean decrease – I mean just decrease the right hand side of [inaudible]?

Instructor (Stephen Boyd):That's exactly what I mean, yeah.

Student: Did you go negative?

Instructor (Stephen Boyd):Yep. You go negative. That's heightening that constraint. And here you're guaranteed – I can tell you that if you tighten U two for this problem, I can give you a lower bound on how bad things can get. You can tighten U two and P star can go to infinity, which is to say it's not feasible. So I can't give an upper bound if I tighten U two. I can give you a lower bound though. If I tighten U two to minus point one, for example, then someone tell me here how much P star is gonna go up by at least.

Student: Twenty.

Instructor (Stephen Boyd):Twenty. Precisely. It'll go up by one. Could go up by plus infinity. If you tighten U two to minus point one, and this thing becomes – the problem becomes infeasible, then it went up by plus infinity. But it has to go up by one, period. So that's the picture. Okay. Now let me also put this in perspective. It's gonna turn out the following. More – I mean, for the things we're gonna look at this quarter, not next quarter, but for this quarter, although duality plays central role next quarter, but when you look at them – sorry, I know you're not thinking about next quarter – as a [inaudible] order statement, the following is true. When you solve a convex problem, you're gonna get the optimal lag range multipliers whether you like it or not.

It's just – that's – you, in fact, will see all this in a few weeks. I mean that's basically how they're solved. I mean – so – and you've heard things like primal dual solvers, or you may have heard of this, or whatever, but it doesn't matter. Even if it's not a primal dual solver, whether you like it or not, you will not be given just X star. You'll be given X star, lambda star, nu star, and that's universal. So any solver is gonna do this, period.

You may not care about lambda star nu star. You may only care about X star. Fine. But the point is they're there. Maybe they're not being returned to you. Maybe they're being returned to you and you're ignoring them. They're there. What that means is basically these optimal sensitivities, if you want to call them optimal sensitivities, they're free, they're free. You design a circuit, period, you get these optimal sensitivities.

You can ignore them. Fine. No problem. But the point is they're absolutely free. They're just there. You know, you solve a maximum likelihood problem, these things are there. You do experiment design, these things are there. You do finance, they're there. So actually that's why, actually, all of everything – all the duality we're talking about actually has huge applications because this is all – this all comes for free.

These are not – there's no additional cost to calculating these. You get these whether you like it or not. So I – seems to me the right policy is when something comes to you for free, you should figure out some way to use it. And they're just fantastically useful just to look at. So this is a matter of course. And if you look at any solver – and I mean, like, a low level solver, one that solves an LP, a QP, and SOCP, and SDP, mathementropy, any of these, absolutely no exceptions, it will return dual feasible points.

I mean that's just part of the deal. Actually, if for no other reason than to certify the solution, right? Because that's what's considered in convex optimization what it means to solve a problem. You don't just say here's the answer I think. You say here's the answer, oh, and there's no reason you should believe me, so I'm also providing optimal lag range multipliers, which in fact form the certificate proving that what I have alleged as the answer is in fact the answer. Everybody see what I'm saying?

So that's just sort of the – that's sort of the social contract you have when you solver for convex optimization. That's what you get. Right? So – and then there's no questioning. I mean there's no questioning. I mean there's no – then – by the way, it's none of your business how they solved the problem because they've returned not only a solution, but a proof, a certificate proving it's the solution, and then it's none of your business.

They could have figured out X star by using a stick or something like that. It – but, I mean look. It's fine. It doesn't matter. If the certificate checks out, that's the end of that. That's the end of the discussion. Okay?

Student: How do you feasibly check certificates in your hand?

Instructor (Stephen Boyd):You evaluate G. So you evaluate G, and you – I mean, first of all, if you really want to – if you want to do this in a way where you don't trust the solver, it's extremely simple. Somebody gives you X – let's call it X tilde. We'll take a tilde off and put a star on it only after we certify it. So they return to us X tilde, lambda tilde, nu tilde. Okay? Tilde because it hasn't been verified yet. Right?

So we're doing safe optimization or something like that, right? So the first thing I do is I take X tilde, and I check if it's feasible. If it's not feasible, I leave a bad review for that solver. I go back and leave some customer feedback and say, come on, this is – they return to me a point that was supposed – they alleged to be optimal, it wasn't even feasible, okay? So if it – if X tilde is feasible, I think check that lambda tilde is bigger than or equal to zero, okay?

If they have a negative lambda tilde, I give an even sillier – I give an even worse review of that solver in a reputation system, okay? If lambda tilde is bigger than or equal to zero, now I go back and I check the KKT condition with the gradients and whatever – the fourth KKT condition. If that derivative – if that gradient is not zero, then I say come on, please. Stop messing with me. Okay? If it's zero, then I know what G of lambda star nu star – sorry – G of lambda tilde nu tilde is.

If that number is equal or very close to F zero of F tilde – if it's within my threshold, it's certified, end of story. That's it. And I don't – I leave positive feedback for that solver. Okay? So that's how you do it. Okay. All right. Next topic is really interesting, and let me explain what it is. There's a question.

Student: So for [inaudible] you just skip the differentiability step? You go straight to evaluating G?

Instructor (Stephen Boyd):There is a direct variation extension of KKT conditions for non differentiable, and you use the idea of a sub gradient, not a gradient. So there is an extension of it, but we won't do it in this quarter. In fact, generally, this quarter, the way things are gonna work is this. You can actually reformulate your way – well I'd say this is a good topic. Sort of like the next topic. You can reformulate your way around most – almost all non differentiable problems, right? In fact, it's shocking the extent to which you can.

I mean let me just – let's look at an example, right? So suppose someone says, well, I want to minimize – note the period – A X minus B infinity. Like that. That is horribly non differentiable. I mean the objective here is piecewise linear. And, you know, whenever, you know, one entry of A X minus B hits the maximum and takes over the maximum, you've got, like, kinks in the objective. This is horribly non differentiable. Okay?

So you say, well, that's hit. KKT, I can't do it. But interestingly, we know how to – you know how to reformulate that. That was all the stuff we did in the last week or two. What you do is you rewrite this as – I mean there's lots of ways, but this one you might write as T – that's a new variable subject to A X minus B is less than T one, and it's less than minus T one, like that. Okay? Now what's super cool about this problem is it now has N plus one variables.

It also has two M linear inequalities. But notice now – totally differentiable. So I cannot apply KKT to this because F zero is non differentiable. I can apply KKT to this. So you see what I'm saying? So for us – mostly, for non differentiability, we – our policy generally, pretty much, for this course anyway, is to reformulate our way around it. So however, having said that, there is a generalization of KKT directly to non differentiable cases. So okay.

Now we're gonna do something that's actually very interesting here. And let me just – let me give you the big picture first. So here's the big picture, and at first it seems ridiculous, so let me just show you what it is. So suppose I have a problem that's P, okay? And we know how to form a lag range dual. We'll call that D, okay? So we have a fixed procedure by which we can do this. Okay? So that's – and you know how to do it. I mean it's – you take the original problem, you form a lagrangian, minimize over X, maximize that, call that G, maximize G subject to lambda – you know, that's it. That's this thing.

Okay. All right. Now what we're gonna do is this. I mean this is just really dumb. Here's what we're gonna do. We're gonna take a problem and we're gonna do an incredibly stupid equivalence with it, and I'm gonna create something called P tilde – and I mean really dumb. Wait until you see how dumb these – they're nothing – this is a sophisticated reformulation. Okay? We're not gonna do that. We're gonna do, like – you'll see how stupid that these reformulations are, and we're talking really simple here.

Okay? Like introducing a new variable and saying it's equal to the old one. I mean it literally – things like that. I mean that's how dumb it is. Okay? So we're gonna do that. Well – so this is completely trivial, and then of course, well, we know we can form a lag range dual like this. Okay? Now aesthetics would suggest that these two problems should be trivially related. That's what it would suggest, right?

And in a sense they are because, for example, D and D tilde are related because they are the duals of primal problems that are trivially obtained one from the other. Okay? And so you might – you want to add this in. I guess when you do this in math it's called a communitive diagram because it means you kind of – either way you go you kind of get the same thing. You want to do this. Well guess what, this doesn't work at all. There's – these can be – they can look wildly different.

I don't want to say they are unrelated. They are related. They're related this way – only this way. But there's no obvious connection between the two, and we'll see stunning cases where the most mind-numbingly stupid transformations of a problem – if instead of forming a lag range dual you first transform it in a profoundly stupid and simple way, and then form a lag range dual, you get something totally different. I mean totally different. We'll see examples here. Everybody see the picture?

Now at first this looks bad because, well, first of all, it's unaesthetic. You want it a beautiful picture with – you want to – well if you're trained in math, you want to draw that last thing. That's called a communitive diagram. Anyway, you'd want it to work that way. And at first, it's upsetting that it's not there. Actually, it's very, very good because what it – actually, it's gonna be interesting because it means – it means that – so there's the idea of weak sort of – not weak, but there's sort of the – you can do the basic lag range dual.

And that's where you just follow the book. Somebody gives you a problem, you form a lagrangian, form dual function, maximize dual function subject to lambda positive. That's, like, by the book, okay? What – this fact that there's not a communitive diagram here says there's actually some art and – I mean there's all sorts of interesting things. It means that if you call – if you're willing to call this – this one you would really call the dual because it's created by just turning the crank – no creativity.

This one – I guess probably the technical term is it's a dual of that problem. Okay? So generally when people say, you know, derive a dual or something like that, it's actually quite interesting now because it allows you to do some transformation – this is – I'm talking about how this is interpreted on the street. Okay? So if you're in a courtroom and someone says find the lag range dual, you do it by the book, okay? But if you're among friends and things like that, and someone wants to find a lag range dual, you can actually transform it slightly, then form a lag range dual, and then someone would say is that the dual? You'd say, well, no, it's a dual.

And in fact, if you look in the literature, that's why you'll often see these duals with different names. So if you read a lot of the literature, which I don't really recommend, but if you were to do that, you would find things like, you know, the Frank Wolf dual, or the, you know, the Smith dual, I mean – you have all sorts of names, and they just follow this thing. Okay. So, all right.

So I've made these points here that equivalent formulation problem can lead to very different duals, and let's look at some examples of it. I mean – so here's the most – here's a dramatic example. Okay. It goes like this. Ready? A non-constrained problem – actually, you know, we haven't had a discussion of the dual of an unconstrained problem, so let's have it now. It's gonna be a very short discussion by the way.

So there are no constraints, so what's the lagrangian? Well, for each constraint, we should add a term, lambda I F I, to this objective, and for each equality constraint we should add nu I H I. But there are no constraints, so you're looking at the lagrangian. That's the lagrangian. And there are no dual variables. Well there's no constraints. All right. Fine. You've turned the crank. The next step is to calculate G, the dual function. How do you calculate – well you minimize the lagrangian.

So you minimize this. The minimum of that is P star, right? I mean – period. Therefore, there's a dual function, you know, it's a bit – I mean, I don't know, you could argue about semantics here because it's a strange function in the sense that it actually has no arguments. But imagining you could pass something to G, which you can't, it would return the answer P star. Okay? So now – let me tell you the good news.

Good news is, well, there's zero duality gap. By the way, P star is an excellent bound on P star – lower bound. As lower bounds go, they don't come better. But it's completely useless [inaudible]. It's utterly useless. All right. Now watch this. I mean look at this problem in this problem. I mean you can't get a stupider transformation than that. This basically says I'm gonna call – introduce a new variable. I'm gonna call it A X plus B. That's all.

I'm gonna call – sorry – it is A X plus B, I'm gonna call it Y. And so I'm gonna minimize F zero of Y, subject to Y equals A X plus B, that's what this is. Now you have to admit, as transformations go – and by the way, this is so stupid that anybody who starts this way, you should think, look, this is so dumb and simple, there's no good – nothing interesting is gonna come of renaming something. I mean that's all this is. It's just renaming something. Technically it's not.

You've actually introduced a new variable and new equality constraints. Okay? The point is this is no deep, you know, this is not a complicated transformation. Okay. Now the dual here however is this. You form a lagrangian and it's F zero plus nu transpose times A X plus B minus Y, okay? That's – this is the lagrangian, and you minimize that, and you get something interesting. You get minus F zero star plus B transposed nu if they transposed nu as zero. And so the dual is this. This – so here the original problem – unconstrained problem had a completely useless dual.

There was – the dual function was just P star. Just a constant. Now you get something that's not remotely obvious. It's this. It's different. And let's just look at an example. I mean here's an example. Minimize norm A X minus B. Well we do that over the time. This is a general norm by the way. It's not too norm. Okay? So just general norm. Minimize norm A X minus B. Well if you just form the lag range dual – if you're on the stand under oath, and you're asked to form a lag range dual, you say, yes, I've done it.

They say, can you tell us what the dual function is? You say no problem. It's P star. They say, what's P star? And you go, it's the minimum value of A X minus B norm. And then there's nothing they – that's not – that's it. That's all you say. You just say that. They'd say that's the lower bound on the optimal value of this problem, which is also P star, so everything's cool. If you apply this procedure, you end up with this. I mean you minimize norm Y subject to Y equals A X minus B, well, I mean, come on.

Transformations don't get simpler than this. You derive the dual of this. You get something very interesting. You end up with the following. You work out what the conjugate function of the norm is, and the conjugate function of the norm is the indicator function of the dual norm unit ball. I think we've actually done that. But what happens is you end up with the following problem. The minute the dual of this – now by the way, this is sort of a lag range dual, so you would – it's not – you can say the lag range dual exists in this – this the lag range dual of that.

Is this – you maximize B transpose nu, [inaudible] A transpose nu equals zero, and nu in the dual norm is less than one in – well in dual mode, sorry. Okay? So this one is useful. It has huge number of uses actually. Tons and tons of uses. It is not obvious at all. Now by the way, the way this would work is this. Some people would – so there's many ways you could say this. You could certainly say this is a dual of that problem, and someone says really?

And you'd say, okay, look, technically, here's what it is. It is the lag range dual of a trivial reformulation of this problem. That would be if they ask you really – because there's actually other ones, and you'd get other duals even for the norm approximation problem, and that's why if you look in the literature you'll see all sorts of different duals, and you’d see the Smith dual, and the Frank Wolf dual, and then this the dual, and blah, blah, blah like that. Okay?

That’s why it is – you could say – you could look up QP duality and find four different duals. Okay? One will be the dual that just turns the crank that we have. Others will be ones where there’s some slight reformulation, and then you form a lag range dual. And they’ll look different. They’ll look as different as these. So anyway, yeah?

Student:[Inaudible] create [inaudible].

Instructor (Stephen Boyd):Nothing.

Student: Nothing?

Instructor (Stephen Boyd):Nothing at all. So here – in fact, let’s talk about duality gap. Let’s – for this problem specifically, somebody tell me what is the optimal duality gap for this problem? Why? It’s actually zero. So in other words, there is no issue. This problem and this problem, same optimal value. Why?

Student: Convex [inaudible]?

Instructor (Stephen Boyd):Yeah, so if Slater holds trivially, Slater says any inequality constraints – for any inequality constraint – so sorry – for the inequality constraints, there must be a point that satisfies them strictly. Well look at that problem. There are no inequality constraints, therefore Slater holds. There’s nothing to show. Done. So P star equals D star. Now since these two obviously have the same optimal value, therefore the optimal value of this one and this one hold. So it works.

Okay. So there’s another trick, which is this – and we’ve already seen this a couple of times. You can actually – there’s a difference explicit and implicit constraints. It doesn’t really matter generally. I mean it’s kind of just a trick in labeling, you know, in general. When you form a dual, it is not a trick. So you will get different duals. So if you include, you know, when you have a constraint, you can either explicitly declare it in your list of constraints, or you can sort of secretly attach it to the function, the objective object, as part of the domain. Right?

Everybody – so, I mean, if you look at these originally, these are just sort of reformulations, and it’s just kind of trickery. But you can imagine sort of, you know, if you imagine, say, software that implement – you can imagine how this might work. They’d actually be a little bit different. So in one case it’s in the domain of the objective – for that matter, you could sneak it into the domain of one of the constraint function or whatever, or you can just have a domain of the problem, but that doesn’t matter. So let’s see how that works

So here’s a problem which says minimize a linear function subject to equality constraints and a box constraint on X. Okay? That’s obviously an L P here. You can write it out as an L P. This is an L P. That’s the dual L P. And you have lag range multipliers for the two inequalities, you know, that lambda one is for this one, lambda two is for this one. Oh – say, this is fun. Let me just mention something since we’ve just covered this. It’s actually quite interesting.

What can you say about the entries of lambda one? Very poor choice. Sometimes you’d call this lambda minus and that lambda plus or something, you know, because it’s the upper and lower bound. Can the entries of lambda – could the third entry of lambda one and lambda two both be positive? And if the answer’s no, explain to me why. [Inaudible] that’s fine. So – but finish the argument. I agree. Finish the argument.

Student:[Inaudible] that’s one and minus.

Instructor (Stephen Boyd):Yeah. So if a third component of lambda two were positive, by complimentary slackness, X three has got to be slammed up against this inequality, and therefore X three has to be one – X three star, right? If also the third component of lambda one is positive, then it says this inequality, X bigger than minus one, has to be tight, and that would tell you that X three has to be minus one.

Now numbers cannot be minus one and one at the same time. So this is impossible. Everybody follow this? So actually, you will see this as actually a practical use for this. It means that lambda one and lambda two actually share – so actually, people will do this, actually, as a data storage – one trick is actually to store them in the same array, and they’ll put a minus sign on lambda to code whether it’s at the lower or upper bound.

And so they’ll actually call this lambda plus and lambda minus. It’s actually – it’s a good trick. It actually makes sense. But you will actually see that. You go to Google and get some codes, you will find optimal lag range multipliers for what people call, you know, bound or range constraints, they will be returned as – in some cases as one vector where it’s reported as positive if it’s the upper bound is tight, and negative if it’s the lower bound. That’s just an aside.

It just had related to what we’ve just done, so I thought I’d mention it. Okay. All right. So this is the dual. Everything’s fine here. No problem. That’s another L P. But now what we’re gonna do is this. We’re gonna rewrite this problem this way, and then there’s no – well, okay, clearly they’re equivalent. By the way, they’re not the same problem because actually you would get different exceptions thrown at you when you pass in X, which violates the – if you give an X outside the box over here, the objective will happily be reported – evaluated.

The objective functional will record its value and pass it back to you. Okay? A X equals B. Well let’s say that’s zero and it’ll give – it’ll come back and give you the [inaudible] token. It’ll return the [inaudible] token. Then you’ll throw X into the box constraints, and one or more of them will come back and say infeas. Okay? By the way, which means that your objective value is interesting but not relevant. Okay?

Now in this problem, if I throw – if I pass in to this problem and X, which is out of the box, the objective throws an exception at me, and the objective throws like an OOD token back at you, which means out of domain. Okay? So that's the difference. Of course, it makes no difference. I mean it doesn't – but now let's form the lag range dual and it'll make a big difference. Let's try it. So here I simply take – the dual function is I take F zero, as usual, plus nu transpose A X minus B, this, that.

And I have to minimize this over – and I've already simplified a bit here. So really this minus one less than or equal to [inaudible] one, to be honest with you, to be fully honest with you, this thing is actually attached to that object. It's an attribute of that object. You see what I'm saying? Because – anyway, I just pulled it out, okay? But I know how to – in fact – wait a minute. I think you've solved one. Didn't you do this on some homework?

This is one of your trivial L Ps. Minimize C transpose X subject to X in some box. Didn't you – I think you did that. I know I talked to people in office hours about that, so – and I think they were in this class. But maybe you did something like that, or something close to that. Anyway, the solution is very simple. You select X to the plus one of C is negative, and minus one of C is whatever [inaudible]. Anyway, if you do that, then the optimal value of this thing is this, it's the one norm of this.

And so you actually just get – sorry. It's not just C. It's A transpose nu plus C like this, and you get that. And that's the dual problem, okay? So here, the originally problem is – and you might, by the way, to make them look more like duals, you might write this this way. Yeah. There you go. So you could say I want to solve – minimize a linear function so there's a linear equality constraints and an infinity norm constraint.

Then – by the way, you might say, oh, hey, here's the dual. It's unconstrained, and you minimize a linear function minus a one norm. By the way, when you write them out by norms, it gives people that you're getting that good duality feeling because you know that infinity norm and one norm are conjugates. And so they're related by duality. So this sort of makes sense. So you would say it depends how well you know someone and what the social context is, but you – it would be okay to say this is – I guess properly you'd say this is a dual of this original problem. Something like that. Question?

Student: How did you get to [inaudible] of X Y?

Instructor (Stephen Boyd):Here?

Student: Yeah.

Instructor (Stephen Boyd):Oh, I collected A transpose nu plus C transpose X, and I worked out this. There you go. Like that. And I know what the answer to this is. I choose X to be plus one if the corresponding entry of A transpose nu plus C is negative. And the optimal value of this thing is the one norm. of A transpose nu plus C. So that's what I did. It's a dual – basic dual norm result. Yeah. Yeah, I'm not expecting you to follow every line here, obviously, I'm going way too fast. I mean you can get most of it, so if you're getting every line, then I'm going too slow.

So – and I – in that case, I presume I would be getting the international non-verbal signs – boredom signs from you. So I should be getting that from about 10 percent of the class, or something like that. Okay. So that finishes up – I mean that was just a quick topic, but it's just to point out – and you'll do examples of this in homework and stuff like that. So – and you'll see it in – you'll get these ideas. But the main point was this, and it's not obvious. The main point is this.

Trivial reformulations of problems, you don't get a communitive diagram of lag range duality. So just – and it's interesting. You can get very different duals by doing things that would appear to be silly. I mean even dumber is, like, if you had to take this norm problem, and you minimize the norm, you could say well, look, minimizing the norm is as minimizing the norm squared, you know, obviously because the norm is positive not negative.

If you then form the dual of minimizing norm squared, you get another totally different dual. Totally different. I mean it's not totally different – it's got the same problem data in it, it's also got a one norm in it, but it'll be different, and they'll have different uses, different applications, and things like that, so okay. Our last topic – we're actually gonna skip a very important topic. Sorry, but I'll tell you what it is. It's duality for feasibility problems. And these are actually called – I mean, but it's in the book, we do expect you to read it, and we're gonna find some homework problems on it I guess, or let's make sure we do.

At least one or two, or something like that. And in this case, you get things called theorems of the alternative, so I'm not gonna cover it because after you've sort of seen a whole bunch of this stuff you can guess it. What I want to do now is – the last topic in duality is problems with generalized inequalities. So here we're gonna look at – so we have F I of X is a vector. Everything works the same, except remember how it works in our case is we form, you know, lambda I F I in the scaler case, but now F I is a vector. So it's not surprising that lambda I should be – has to be a vector too, and this becomes [inaudible]. Lambda I transpose F I. That's fine. Here's what's cool though. Lambda I for scalers were bigger than or equal to zero. Now there's – for scalers, there's no very many interesting inequalities other than the ordinary inequality, okay? That's it. So for vectors, you have lots of notions of what it means for lambda I to be positive, and it turns out – and you've got – by the way, it had to be for aesthetic reasons. For various reasons, it had to be – lambda I has to be positive in the dual cone. Okay? And if you check carefully, it even sort of makes sense. So here's what it is. This is the lagrangian. If you look at – so F I has to be less than or equal to zero in the cone K I. If you look at sort of dual cones and everything, you can actually mark each thing as actually sort of being in the primal or dual space.

We don't generally do that. I mean that would be distinguishing between row vectors and vectors, or, in a mathematical context, linear functionals and vectors. So we're not doing it in this abstract way, but you can do it that way and it will work out. And lambda I transpose has a row vector. It's a linear functional and it has to be the dual. It can only be the dual cone positivity. So – and everything else works. You have the lower bound property. If these lambdas are bigger in the dual cone – bigger than zero, then G is a lower bound on P star. And actually, everything just – it's the same. I mean these are zero, you know, and then this is a product of elements in the cone – that's in the negative cone, and that's in the dual cone. And inner products of things in the dual cone and the cone are non-negative. Inner products of things in the dual cone and the negative dual cone are non-positive. And so this thing's less than or equal to zero. Everything works. Actually, everything works, including Slater and everything. And I just want to do one example of that. It's a semi-definite program. So you want to minimize C transpose X subject to – by the way, this is called an SDP in inequality form, or something like that. Subject to an LMI – that's a linear matrix is less than or equal to, in matrix sense, another one. And the lag range multiplier here is actually gonna be an element of the – I have an inequality – it's gonna be an element. Also, it's gonna be a matrix inequality. So it's gonna be matrix Z symmetric. And it's gonna have to be non-negative in the dual cone.

Now the dual of the positive semi-definite cone is the positive semi-definite cone, so Z's gonna end up being a positive semi-definite matrix. The inner product between two symmetric matrices is trace – in general, actually, it's just trace A transpose B. But if it's symmetric, it's just trace, you know, A, B, period. So it's the trace of this thing, with a minus sign there, times Z. So that's the lagrangian right there. And you stare at it for a long time, you realize it's affine in X. We know how to minimize an affine function. It's minus infinity unless the linear part vanishes. And so you get this. It's minus infinity unless these things – that's actually that X I component of the linear thing here – unless that is equal to zero, in which case, this all drops away and what's left is the trace Z G. That's this guy. And so this is the dual STP. Okay?

Now by the way, this is not remotely obvious. We didn't use anything fancy here. This is just lag range duality for – it's not immediately clear that these two are intimately connected – these two STPs. By the way, this is an STP in so called equality form. And depending on a person's background, they'll either think of this one as the primal and this one as the dual, or this one is the primal, this is the dual. So I think if you're working optimization, this is the primal and that's the dual. If you work in control or signal processing, that's the primal and that's the dual, but these are just – obviously it's symmetric. Okay. So we will quit here. This actually finishes, essentially, all the base theory for the class. We're done. Now we'll do a lot of applications and stuff like that.

[End of audio]

Duration: 79 minutes

ConvexOptimizationI-Lecture10

Instructor (Stephen Boyd):Well, today we're starting, well, a whole new section of the course, although the boundary's not that sharp. The next sort of whole section of the course, and book, for that matter, is applications. So we'll just look at a bunch of applications. Instead of looking at sort of the analysis and things like that, we'll look at applications.

After a while, it gets a little bit tedious, because you start realizing in some ways, a lot of them are the same. But we'll look at a bunch of them, and you'll do homework on a whole bunch of them, that'll give you the basic idea of how these things go. And in some sense, this is really what the class is about. It's about actually formulating and using these methods for problems.

Okay, so our first topic is going to be approximation and fitting, and we've separated this so that at first we're not going to talk about statistical methods but we'll get to statistical methods actually very soon. So our first treatment of approximation and fitting is just non-statistical. But we'll get there and we'll see that it's actually very close. It's actually very interesting, the connection.

So the first topic, if you go down to the pad, is normal approximation. So this is a problem you've seen, absolutely certainly in the case of – with a 2-norm here, and it simply, you minimize the norm of AX minus B. So that is the problem. You choose X. And this has lots of applications, so geometrically you could say something like this: you are computing the projection of the point B on the range of A in this norm.

Now actually, we'll see that the choice of norm is going to actually be quite interesting here. There's lots of choices here. So that's the geometric interpretation of this problem. Another one is estimation. So you have Y equals AX plus V, and V is some kind of measurement area that's unknown, at the moment it doesn't have a statistical model. But the point is it's sort of assumed to be small. In fact, specifically, it's assumed to be small in this norm.

So the norm of AX minus Y in this case is in fact the norm of your imputed V, because if you guess an X and you've measured Y, then for all practical purposes you're saying that X minus AX is what the noise was. And then the norm is a measure of plausibility. Actually, it's a measure of implausibility of the noise V, right?

So for example if norm V is large, that means it's highly implausible, your guess. If the norm is small, it's more plausible. So that's the idea.

Here, the best guess is going to be X star, the one that will maximize plausibility here. In another case, it's just optimal design. So in that case, the vector X or its coefficients, these are design parameters. These could be inputs in an optimal control problem. They could be anything. They could be inputs that you're going to send to a communication system, something like that.

AX is the result of your action, so X is an action, and then B is something like a desired action. And here, the question is to find the – sorry, the desired result. To find the action or inputs so that the actual output, that's AX, is as close as possible in this norm to this desired outcome B. So that's – all of these are just applications of a minimized norm AX minus B.

You should have seen examples of this. Least squares, I mean, certainly you've seen, or regression. So least squares just says minimize this in the 2-norm, this has a solution. In fact, the optimality conditions are exactly this. It's A transposed to A, that's the normal equations, X equals A transposed B. So if X satisfies this, it minimizes norm AX minus B, and actually, that's if and only if. So that's the condition.

If – this actually always has a solution; in particular, A dagger B, where A as the pseudo-inverse will work. But if A is full rank, then A transposed to A inverse, A transposed B will work. So this – and this is just review.

[Inaudible] approximation says it will measure the norm, in fact, by the maximum of the absolute values of the – this is called the residual, X minus B. So we'll look at the maximum of the absolute value. That, of course, transforms to an LP, so that's nothing but this. It's minimize T, T is a scaler variable subject to these two M linear inequalities here. There's one here and one here.

Another possible norm at the other extreme would be the 1-norm. in the 1-norm, you minimize the sum and the absolute residuals approximation, and that can be solved again as an LP. Here, you introduce a full set of M separate new variables, and you solve this linear program.

Okay, now these are – it's actually useful to look at this from a – even just go all the way to a more general form, which is not even a norm. So you look at a penalty function. So the idea is this: you want to minimize some penalty function, a sum of penalty functions of the residuals.

So AX minus B is called the residual. It's – well, it's just how much you're off in solving AX equals B, that's the residual. And of course by choice of X you can shape this vector, and in fact your possible choices of R is an affine set, right? It's the set of all AX minus B where X ranges all of our end. That's an affine set.

So you have this affine set, and you want to choose – you want to minimize the residual – some penalty of the residuals like this. By the way, even this is a specialized case, because here we have the same penalty function for all residuals. You could obviously care much more about some residuals than others, but once you get the idea, it's easy to generalize this any way you like.

Okay. So this is a penalty function approximation problem, like this. And you can even think of it, if you like at this point, you could even think of it as a multi-criterion problem, and here we're just taking a sum of them, but anyway.

So here are some typical penalty functions: quadratic. In fact, that's kind of the most typical one. That's for several reasons, which are all connected and tied together. First of all, historical. It's probably the first penalty – the first widely used penalty function, and actually currently most widely used penalty function in many applications. It's just quadratic. Why? Well, because there's an annalistic solution for it, like this. So – and these are obviously not unconnected, right? So that's that.

Student: Could you parse the objective function? What you mean by evaluating [inaudible] all of these different values of R?

Instructor (Stephen Boyd):Well, you commit to an X. You say, here's my X. I calculate the residuals. You got M residuals, and you run up a bill. For each residual I apply my penalty function. That's your total bill.

Student: You have to use the same penalty for each?

Instructor (Stephen Boyd):You don't have to, but we are here.

Student: Oh, okay.

Instructor (Stephen Boyd):Yeah. Okay. So this is just one. I mean [inaudible] it's the most widely used for many reasons and so on. Now there's other options, and we've seen another one which would be sort of in an L-1-norm would give you this – something like that. And of course you could have things like the three halves norm and things like that, if you wanted to.

So another one – but others are more interesting. One would be like a dead zone function. That's a function that looks like this, and then for example grows linear. That's this function. And in fact, the way you should conceptualize the function five, the penalty function, is extremely simple: it's a map of irritation versus residual. So it's how irritated you are with the residual of a certain level.

I mean, I know this is silly, but it points to the square function, tells you that if a residual is small you're actually irritated very little, which is small squared. If a residual is big, though, the square is very irritating, meaning it's big squared. If you have something like the one norm, these two reverse, relatively speaking. If you have a small residual, you find it, relatively speaking, very irritating. I mean, compared to a square.

If it's large, you find it relatively less irritating. So these are stupid ideas, they're very, very elementary, but it's actually all gonna come together. It'll actually all come together with maximum likelihood estimation and statistical models, and it'll all sort of make sense.

But the first thing is just to think of it really that simply. If you have a dead zone linear model, what you're saying is the following – I mean, it's so dumb it's amazing. It basically says for example here, I don't know what this – where the threshold is, maybe

.25 or something? It basically says the following: it says that actually for residuals between plus-minus .2, I don't care at all. That's good enough. For residuals bigger than that, I start carrying, and I start carrying linearly, okay? So that's all.

Then you can go the other way. You can have – for example, here's a log barrier. This is minus A squared log, one minus U over A squared, and if you work out what this is, it's actually very, very simple – this thing will go to plus infinity, outside the open interval minus AA, and so that's [inaudible] here, that's a log barrier. By the way, the log barrier will coincide with very high accuracy with a quadratic for small A. So this looks exactly like – in fact, I mean, it's very, very close to simply U squared for U less than let's say .3A or something like that – .2A for sure. It's really, really close.

But then this – what happens is this says that for small residuals, you care sort of quadratically. But as they start getting bigger, you care super-quadratically, and in fact you are, in this case, infinitely offended by residuals that are bigger than or equal to A in absolute value. So that's the meaning of this one.

Well obviously when you solve this problem you get different values of X and different residual distributions. So here's just a silly example just generated – everything's completely random, A is 100 by 30. So you have 30 variables, and you minimize the penalty function, penalty applied to the residuals for – this is absolute value, this is going to end up being the same as 2-norm, dead zone linear, and a log barrier.

And here's what you get – and actually, this is already really interesting. Let's go back to quadratics. So that's the quadratic residual, and you see a lot of residuals sort of packed around the value – let's see, that's a half? You know, plus-minus a half. You see a lot of residuals here. They're kind of evenly distributed.

You see a couple of residuals out here, okay? Now, these are out here because they have to be out here. In other words, you can't have all the residuals small or something like that. That's actually the point. If you look at the L-1 solution, you see something much, much more interesting, as you can look at the scale to sort of figure this out.

There are a whole bunch of residuals that are – well, a histogram would tell you near zero, but I'll tell you exactly what they are: they are zero. So there's a whole bunch of residuals; in fact, something like almost 40 – 35 are exactly zero. That's what this is, okay? Now, how can you explain that? Well, if you compare it to, say, least squares like this, it's – I mean, we're gonna explain this many, many ways, but just to get the first intuitive picture of why this happens, this is quite easy.

I'll anthropomorphize the solution of these convex problems as nothing more, but here's what happens. In L-2, once you get a residual small, you expend no effort to make it any smaller, because your irritation level is small squared at that point, and there's just no point – you don't care. So the importance is that once these residuals are pushed into sort of in this area, there's no – the diminishing marginal irritation is very small, and so you just quit, okay? That's fine. You focus on big residuals.

Now in L-1, it's actually quite interesting. I mean, this is kind of – everything I'm saying is kind of obvious, but you'd be surprised at what the utility of some obvious things are. The marginal irritation in L-1 is constant. In other words, decreasing something by .1, if it's out at 10, or if it's at .1, is actually about the same. Well, sorry, it's not about the same; it's exactly the same, right? To decrease a residual by – so your marginal irritation level, that's the derivative of the penalty function, is constant.

So what this says is you will – when you have small residuals in L-1 – and again, I'm anthropomorphizing it, but that's okay – it's actually worth its while to actually take those residuals and push them all the way to zero. That's the first – that's the reason you actually get a whole bunch of zeros here.

By the way, this is actually very – this is sort of – this has been around, people have known these things and used them for, I don't know, 20, 30 years now. But in the last couple of years, ideas related to this are very, very fashionable, let's just say. They're not just fashionable; they're actually useful. But they also happen at the moment to be very, very fashionable. And I'll talk more about these in some other context, so. The point here is the residual distribution for L-1 is you get lots of zeroes, and that is not an accident by any means.

All right, let's look at dead zone linear. Well, dead zone linear basically says – yeah, that's minus a half, so plus-minus a half here. Okay, so what this basically says is there's a free ride on residuals out to plus-minus .5. Above that, you're gonna pay linearly. Now interestingly – I mean, this is hardly anything you – you wouldn't suspect anything else, but of course if it's a free ride for a residual at the plus-minus .5, it's hardly surprising that a giant pile of them will end up right at the boundary of where you start paying.

So they'll end up right at plus .5 and minus .5, like that, okay? Those of you in communications will know that this – or will connect this actually to something called blind equalization, so – or maybe – well, I'm not sure, okay, so, if you're in communication and know about blind equalization, you would recognize this.

So this is hardly surprising, right, that you sort of push some out where it's a free ride. By the way, these are in here sort of accidentally. I mean, because it's completely – these are free to move left, right, doesn't make any difference. They're there just because they're – it didn't matter. It actually helped to have them in here, and it helped out other things.

Out here, you sort of care less about these, and you can see actually that the outliers, for example, are a little bit farther out than in least square, it's not surprising. Okay.

Here's the log barrier. The domain of the log barriers is plus-minus one. So obviously, all residuals have to be between plus and minus one, strictly. I mean, obviously, and you can see that's done. Now they go pretty much up to – almost up to one. Now the reason out here, these residuals are very irritating. They're there only because they sort of have to be there.

By the way, I didn’t show it here but if I minimized L-infinity norm, what do you think the residual distribution would look like? What do you think?

Student: All of them about equal.

Instructor (Stephen Boyd):What's that?

Student: All of them about equal.

Instructor (Stephen Boyd):Equal, or in absolute value?

Student: Yeah.

Instructor (Stephen Boyd):Yeah, exactly. If you do L-infinity norm approximation, then you take the infinity norm, and the infinity norm is simply the – you imagine something that goes like this and squeezes the residuals. And you squeeze the residuals down until you can't squeeze any more, and you're absolutely right – if you do L-infinity norm minimization – [inaudible] approximation, you'll find a whole bunch of residuals will be at the positive limit and a whole bunch at the negative limit. That's exactly right, okay? So that's the picture. And these are all very simple ideas.

Here's one that's quite interesting. It's the Huber penalty function, and it's a function that looks like this. It's quadratic out to some transition point, and then above that it's linear, okay? But we'll have a statistical interpretation – one statistical interpretation of this soon.

It's actually very interesting, what it does. Obviously, it's convex. You can do a Huber function approximation. What's extremely interesting, it sort of combines L-1 and L-2, and let me show you what it does, and you can sort of guess. So here's an example. I mean, it's a made-up example, but it gives the idea.

Here's a whole bunch of points – like, I don't know, 40 Q points, like this. Actually, this is 40 points here that kind of lies on the line, and then we threw in two, you know, serious outliers, like that. Okay? Well, the least squares fit is this dashed line, and I mean, it's kind of, you know, obvious that that's gonna happen. That's an extreme outlier, and the cost of one point being way far out is, well, it's way far squared in least squares.

So this thing will actually, even though that's a mere one, two points here, this thing will actually rotate considerably, even though it's just one point, one outlier here and one here. So I guess they're putting a strong torque on it, here. Well, by the way, that analogy is actually quite perfect if you attach to this line springs. Then – of unit, if you put springs with unit stiffness, it's in compliance. It's exactly correct, right?

So these two actually put a torque on it, because they're quite extended and they put a big force on it, and they rotate this like that, okay? That's the least squares thing. Well of course; I mean, that's what you'd expect.

If you make that – if you look at the Huber estimate of these, you get something very interesting. It's the dark line here, okay? And you can see, is it twisted? Little bit. But actually, not much. Now, you know, obviously if you have data, you know, that you can plot, you don't need any fancy methods. You should use your eyeball to fit things.

So obviously this example is not the point. The point is that you can now do this with, let's say, a thousand measurements in estimating 400 parameters or something. And let me tell you, your eyeball cannot possibly identify outliers in things like that. Totally out of the question. Completely out of the question.

This will work in a way that actually often is spooky, it's so good, okay? And we'll see reasons why and things like that. By the way, you can do all sorts of – you can imagine all sorts of methods for treating outliers that go beyond this, so for example, you might use a Huber estimate to start with, then simply go back and look at the residuals at that point.

In this case, it's embarrassingly simple – these two points would be obviously flagged as super-high residual, and then you'd trim the residuals and you'd refit. So you'd flag those data, those measurements as flawed and you'd refit it. And by the way, in this case, it would just work perfectly.

By the way, if I were to crank these down like this, this method in least squares would fail. It would actually start failing. You'd start identifying points up here as the outliers and things like that. The Huber one would actually work for a huge – I mean, when it actually starts getting not too obvious here.

Student:[Inaudible] will actually choose them?

Instructor (Stephen Boyd):Not – for this lecture? Yeah, I can. You do it the way you think you might do it. You wiggle M until you like what you see. Now, no one will ever tell you that. No one will admit that, but that's the truth. That's how you – that's more like asking somebody how do you set the weight functions in regularization? Truth is, they wiggle it until they like what they see.

Now later, actually, there's a very good statistical method you can actually optimize actually by convex optimization over M and X. It's called concomitant something or other. I don't – we'll get to that.

You know. No, yes?

Student:[Inaudible.]

Instructor (Stephen Boyd):You can do cross-validation, that's a perfect – so actually, there are better methods. Oh, by the way, we'll also – if you have any idea about the variance of the nominal noise, then of course that sets M. That would set M right off the bat, so.

Student: For the histograms, wouldn't you think that having a steeper barrier function move your residuals closer to zero rather than spreading them out?

Instructor (Stephen Boyd):Yeah. That's what it does.

Student: Why is it then when you steepen it to the point of infinity, it spreads it to basically uniform? I mean, in the previous slide, it showed, like, pretty much all the bars being [inaudible] uniform as you steepen that curve.

Instructor (Stephen Boyd):Right. So you're not steepening it. Quadratic is much steeper out for big numbers.

Student: Right, that's what I'm saying.

Instructor (Stephen Boyd):So you're making it soft. No, no, it's the other way around. When you go to L-1, you're much more relaxed about big. No one likes outliers – I mean, no one likes large residuals, okay?

Student: Yes.

Instructor (Stephen Boyd):But among convex penalty functions, very important, among convex penalty functions, a linear tail is the most flexible you can be about large residuals. So that's why some will spread out, and it'll allow a few to spread out if you can make up for it with the rest. And that's what this does. So that's the idea.

Also, I don't know if – I'm not sure how this – I'm not sure if the point was heard, but we heard from some of our statisticians that you could choose M of course by cross-validation, so. Which is true. We'll get to some of that. Okay.

Now the dual of these are least norm problems. So in a least norm problem, you minimize norm X subject to X equals B, and lots of interpretations of this. I mean, one is that you have a set of solutions – AX equals B – that's again an affine set, and you want to compute the minimum norm point; that's the projection of the point zero onto this affine set.

In estimation, here, the model is this. You have not enough measurements to identify a parameter perfectly, so I have B equals AX. A is fat here, so. However, the measurements are perfect. So the bad news is I don't have enough measurements. The good news is the measurements are perfect, okay? So I have perfect measurements, but not enough of them, and what that says is AX equals B is the set of parameter values that are consistent with my perfect measurements. So that's what that is.

If someone says to you please estimate X, the answer to that point is I cannot possibly; you have to combine that with prior information about X to say anything intelligent about X. Because, in fact, any X that satisfies AX equals B is consistent with your perfect measurements.

So norm X then becomes a plausibility measure – implausibility measure. So if X is – if norm X is large, that means it's sort of implausible. You assume it's more likely that X is small than large. So that's – and then this says find me the most plausible X or something like that.

In design, you can think of AX equals B as a set of M constraints, or specs, actually – specifications, requirements, whatever you want to call them. There's design choice. In other words, A is fat so there's lots of solutions of AX equals B. For example, X might be a set of forces or something that you're going to apply to something, and this might be some moment constraints or the constraints that a vehicle arrives at a given state at a given time, or something like that, that's what this is.

But there are lots of force programs, let's say, that do the trick, and among those you wanna find the one of least norm, for example. And that would be something like a most efficient design, or this could be a minimum fuel problem, for example, if this was a one norm, and if the one norm was a good measure of fuel usage. Okay.

Of course for least squares, this has an analytical solution, which I won't go into much. In the L-1 case, it's actually quite interesting, because when you minimize the 1-norm of X subject to AX equals B, you get a sparse solution. You get a sparse solution of a set of linear equations.

By the way, there are now – I don't know, every three days, I would say, maybe more, a paper appears on this subject – this topic. So, and literally this one problem right here. And you hear oh, boy, is it fashionable now, and people are talking about – they call it compressed sensing, and you hear all sorts of various names for this.

Very, very fashionable. Actually, in some cases, for some very good reasons. What this does – I can give you a quick example here. A quick example would be something like this. Suppose I want to estimate a vector X. Let's say it's an image or something like that, or it doesn't matter. But I'm told it's sparse. So I have a vector in, I don't know, in R-5000 or something like that, but I'm told it's sparse.

Most of the entries are zero, so it's sort of an image with lots of zeroes, and there's some sparse thing in it, like that. And then I say, well, I'll give you some measurements of X, and these could be, you know, projection measurements, they could be anything. A could be a convolution operator that smears it so I can give you a smooth image. A could be a Fourier transform – this is exactly what would happen in medical imaging. In, for example, MRI, I would give you a Fourier transform of what you're imaging.

Now if B is – if A is square, then of course, I mean, there's nothing to say. You'd just take the inverse – you'd take X as A inverse B. If A is tall, you've taken more measurements than there are parameters to estimate, and of course you'd do something like a least squares, and you actually use the extra measurements to blend and give you a better estimate.

But now let's go down to the case when A is fat. So I want to measure a vector in R-5000, but I only give you 500 measurements. Now of course if someone asks you this, you should normally just turn around and walk away, because it's ridiculous. No one should ask you to estimate a vector in R-5000 based on 500 measurements, right? Well, unless they tell you there's more to it, okay?

So the more to it is that X is sparse, okay? Now that's actually a hard problem. For example, I might tell you X has only 300 non-zeroes. Now by the way, if someone tells you which non-zeroes they are, it's easy again. Because if they tell you which of the 5,000 entries, which 300 are non-zero, I simply pull out those columns of A, I get a reduced system, and now A is skinny again and we have – we're back in the realm of more measurements than parameters to estimate, okay? And everyone's happy.

But they don't tell you. They say, no, no, I don't know. I don't know what they are. But there's only – there's no more than 300 of them. Well it turns out this – if you solve this – this is convex, so it's quite simply, this is quite straightforward to solve – if you solve this problem, then at least there are actually now – first let me just say the practical fact.

The practical fact of the matter is that you will do stunningly well at actually getting the exact X. I mean, shockingly well. So that's actually been known for a while, but that's okay. What is actually new is that there are actually some results that tell you, depending on A, depending on M, the known scarcity of X and so on, that this method will actually, with extremely high probability, produce the exact answer, okay? So this is the idea behind sort of compressed sensing.

So in compressed sensing, in MRI you do it – you run it for one-fifth the time, so you're actually, you have one-fifth the number of Fourier transform measurements than you would normally take. And then you use something like this, so. All right, that was just a little aside, and we'll – you'll see bits and pieces of this in other areas. Okay, so that's just a 1-norm.

And the extension is least penalty, and of course same ideas hold. Except here, the penalty's on X. So for example, if FI is something like a fuel use function, then this would be a minimum fuel control problem, for example. Okay.

Closely related is this idea of regularization. We'll look at a couple of examples, and then you should be able to generalize it. Oh, there are lots of things I'm not saying here because I assume that in the fifth week of the class these are totally obvious. If I go back to these approximation problems and minimum norm problems, I can add any convex constraint I like.

I could say, for example, X is positive, I can give you lower and upper bounds for X, I can give you polyhedral constraints – anything I like. But I don't need to say it, because totally obvious, okay? So that's not even worth saying. So you just see the basic one here. Okay.

Let's look at regularized approximation. Well, the basic idea in regularization – I mean the most basic one, and we'll see extensions of it in a minute – the most common use of regularization is this: you want to fit something or minimize some norm, AX minus B here, but at the same time, you want X to be small.

This comes up in all sorts of things, so it's a bicriteron problem, and in fact the right way to say it is this: it's minimized with respect to R plus two. That's the cone. This pair of objectives. And the Pareto optimal curve, or the optimal trade-off curve between fit and size, if you wanna call it that, is something that would tell you exactly how, you know, these two would trade off. Okay.

And, you know, this has lots of application. I mean, for example, in estimation it would tell you this. It would say there's many ways you would actually – reasons you would want X small. Now remember, to make X small in general you will give up fit. So what you're saying here when you solve this, or choose some Pareto optimal point is you're going to accept a larger – a worse fit for a reduction in size of X. That's the meaning here, okay?

So you might ask why would you do that, and there's lots of things – lots of reasons people might do that. One would be that your model, Y equals AX plus V, is actually only – that's irritating. What do you think? Think I can – I'm just not gonna – I'm not gonna be able to do it. I'm not fast enough. I'll try, though. We'll see what happens. Where is it? Okay. Mm, okay, all right. I'll just ignore it. All right, all right.

So you might know that this model is only valid for X small, and in fact YA could be obtained, of course, as – there could be a nonlinear mapping from parameters to measurements – extremely common. And A could be a linearized version of it.

By the way, A could be from the Jacobean. That would be if you're still an adherent of classical calculus. But A could also be from something like a particle filter or something like that, okay? And to do that, you would of course generate a bunch of Xs that are plausible, run it through your measurement system, get the measurements, and then you'd actually fit a linear function to it, period, okay? So that would be what that is.

And then here you want it small, because if you choose an X that's big and have a very good fit, you could say wow, I have a great fit. But the point is X is so big that your model is no longer good, so your fit is only sort of on paper here. Okay.

In optimal design, you're trading off something like fit versus some kind of cost or something like this. It turns out also, and we'll see this later, that the size of X is related to the sensitivity in this problem with respect to uncertainty in A, in the matrix A. And let me just explain that very, very carefully now, or roughly now and then we'll get back to it later.

Let me ask you this. Let's suppose that A were to change, like all the entries were to change a little bit, like 5 percent. If X is zero, how much difference does it make? None whatsoever. And if X is huge, how much difference does it make? How much? A lot. So I rest my case. So roughly speaking, the size of X is at least related to the sensitivity of AX to changes in A.

So therefore, you might want X small because that's related to how sensitive AX is to changes in A. this is kind of obvious, but. Okay.

So how do you solve a regularized bicriterion problem? You scalarize. And in fact there are many methods to scalarize. This is, by the way, not how it's done, because not enough people know that it's even possible. The number of people who even know you could do this is very small. It includes you and other graduates of this class, but not many others.

So one way to do it is to just add a [inaudible] you scalarize. You take the two objectives like this. Now here, as you change gamma, of course you have a – in fact, what they call it is a regularization. In general, it's called a Pareto optimal curve; it's the optimal trade-off curve of norm-X versus norm-X minus B. It's also called, in this case, specific case, it's called the regularization path. So that's a very common name for it – regularization path, like this, as you vary gamma.

Okay. Now – by the way, the way, depending on historically what the norms are, if you square AX minus B and norm-X, then this actually can be solved by just standard least squares. So it's common to do this. And of course you get the same trade-off curve. It's a different parameterization and obviously gamma and delta don't correspond example, but they trace exactly the same curve out. So every point on this curve is obtained by this – okay, now that's irritating. Don't – don't say anything. No.

That was irritating. It's taunting me. It's taunting me. Hm. Okay, all right. All right, so this is the most classic regularization. It's taken off regularization, and why? Because it stays with least squares and everything's the same. This is just completely standard, so I'm not even going to go into it.

But we'll do an example, and I should say this is only the simplest version of regularization. In general, you tack on – you may tack on a bunch of regularization terms. That's even more common. So for example, I don't know, take an image. You might – it still there'd be no reason to believe that your image is sparse or whatever, and I mean, I don't know, unless you're taking – if you're imaging something from outer space or something like that.

There'd be no reason to believe. More likely [inaudible] it's smooth, and then you'd put something like a smoothing [inaudible]. It'd be the norm. You'd regularize – to make something smooth, you regularize with a smoothing operator. And you could do both, so.

And in fact the way these things generally work is you'd keep adding regularization operators and things like that, regularization terms and twiddling with the knobs until you like what you see.

Student:[Inaudible] is it just like a [inaudible] problem?

Instructor (Stephen Boyd):Which one, here? It is, yeah. It is exactly that. So yeah, it's exactly that, yeah. By the way, that's another way to solve this, is to do this. Is to minimize norm AX minus B subject to norm-X less than some, you know, kappa or something. And then you vary kappa. And the indeed, the Lagrangian for this is something like that. Well, except for a term minus gamma K. So yes, it's related to that.

Actually, you should have build neural links between regularization in – scalarize in multi-criterion optimization and duality, because both of them involve kind of the same picture, and the picture kind of looks like this. Like that. That's the picture. And you change, you know, either dual variables or weights, and this thing rotates and the point of contact rotates. So kind of the same pictures should work in both places. You look confused. No? Okay.

So okay. So let's look at a – this is sort of a typical example of how you'd use regularization. So we have a linear dynamical system, so we just have a convolution operator here, like this. And we have three objectives, so this tracking error, so there's some desired thing and we want the output to track this thing.

So if you only cared about this, you might call that a deconvolution problem or something like that, because that's a convolution and essentially, you wanna unconvolve the desired to get the optimal input. So it's just a straight deconvolution problem.

On the other hand, we don't want U to be gigantic, so we're going to limit the size of U. And we also want the input to be small – sorry, smooth. And to make it smooth, this is an example where we just take a quadratic form, this is a first-order difference, and it's a quadratic measure. Everything here is quadratic. That's it.

Okay. So we used regularized least squares. We're gonna minimize J track plus delta, J, and this is the derivative, I suppose. And eta times J times the – that's really, really irritating. Okay, sorry.

Of course, people watching this later will have absolutely no idea what's going on. It's a student who's taunting me, just for those of you watching this on tape. Yeah. If I get their student ID number, they're in deep, deep trouble, so okay. All right.

So okay, and this is just a least squares problem or whatever, and it's just to kind of give a rough idea of what happens. So in the first case, we take – let's see, so in the first case we put no penalty whatsoever on smoothness, and we put a small penalty on size. And this is the input that we're told does it, and you can actually sort of see here both things are – the two things shown here are the desired one, the desired trajectory you want to hit, and the trajectory you do hit.

And they're not perfect, by the way, and of course if I take eta and make it smaller, these will get bigger and I'll get better tracking here, okay? So that's the first one.

Now in the middle, again, no derivative penalty, but we increase eta. So you increase eta and it says basically I want a smaller signal. And you can see this thing goes between plus five and I don't know, minus eight, or something like that. You say, I'll take a smaller signal but I'll – well, you have to accept worst tracking error, okay? So, and in fact that's exactly what happens here.

You can see it smaller here – well, you can just look at the scale and see that it's smaller. And indeed, you can now actually see worst tracking error. Okay.

And in the final one – in the final one, we take the following one. We're gonna actually now increase a derivative penalty, and you can see here's a smooth input, and your tracking error is – actually, this is the kind of thing you want, by the way. This is what you – what you want regularization for is exactly success stories like this, where you get something you want while giving up very little for it. I mean, that's kind of what – that's why you use regularization.

And in fact, strangely, that's often the case that that works out this way. And here you can see I don't know what the difference in tracking error between this and this is, but it's not – whatever it is, it's not huge, and I'm using the eyeball test here. It's not huge. And yet you could say, again, depends. I mean, you'd have to be able to argue that you like this thing a lot better than that, or something like that.

And you can say look, here's something a little bit worse than that, but look, it's much smoother and it's smaller or something like that. Okay, so this is just this kind of regularization. That's what it looks like.

Signal reconstruction. So this is a very special case in fact of this, but it's widely used, and it looks like this. Here's what's happened. I want a signal. There's a signal I'm given, but it's been corrupted, and just additively, so the linear operator A is I. It's the identity. So I'm giving it corrupted signal, and I want to sort of decorrupt – I just wanna subtract off the corrupted part.

So xhat is gonna be my approximation of what the true signal is, without the corruption or noise or whatever you wanna call it. So when I guess xhat, then I'm really guessing that the corruption or noise was xhat minus X corrupted. That's the Y, that's the observed one, okay?

Now, then if I take, for example, a 2-norm measure of this, this is – often, people would take an RMS measure, that's just a multiple of that, and what that tells you, that tells you, like, how big the noise – your imputed noise is, because you're basically imputing a noise in this case. And we'll see later, statistically this would correspond to a log likelihood term for the noise.

But what this says is, you know, basically if this is small – in fact, what you'd really probably want is you'd want this thing to be on the order of what you would guess it would be if you knew something about the noise involved. So that's what you would really want this to be. If you had some idea of how big the corruption is.

Okay. Now you're gonna pay – you're actually gonna deviate from what you saw, but what you're gonna do is you're gonna at the same – you're gonna minimize, or here's what you're gonna gain, so you're gonna gain a much smoother signal. So FI here is called, like, a smoothing function or a regularization function, it's got all sorts of names, and examples would be things like this. So this is just in one dimension. In one dimension it would look something like this.

A quadratic smoothing function would be this. So it'd be the L-2 norm, squared, and some of the squares of the differences, okay? So this just penalizes the sum of the squares of the differences here. This would be completely classical.

One also very interesting is total variation norm. It's not a norm. People say norm, though, but it's not a norm. So that's – put that in the – flag that with the same flag you use when people say overcomplete basis, by the way, because – anyway, for hundreds of years in math, a basis was by definition not overcomplete. But anyway.

So this is just – it's called total variation, you will hear it called total variation norm. It's not a norm. By the way, it's not a norm because when are both of these – when do these vanish? Or what Xs would they vanish?

Student:[Inaudible] constant.

Instructor (Stephen Boyd):Yeah, constant. I mean, when X is constant. By the way, they all vanish with a vector of all ones, okay? Norms don't do that, you know, by definition. Still people call this the total variation norm.

Anyway, let me show you what this is. Actually quite interesting. If you have a signal, you know, it looks like this. I don't wanna make it too complicated or it's going to be hard on myself. So here's what – the way you calculate the total variation is actually very interesting. Here's another way to do it. It's exactly this: is you identify the peak to valley things, like this, and it's the sum of these – of alternating peak to valley heights. I don't know if that makes any sense. This make sense? So that’s what it is.

So if you see a signal like this, it's the – and so by the way, this puts a huge penalty on wiggling, right, because if you wiggle like this, you run up a very big total variation bill, okay? But by the way, that's also true of the quadratic one. The one interesting thing about this one, which now you can guess because you know a little bit of intuition about these L-1 things, what this says is when you do have to make a shift, total variation is gonna care a lot less about it than least squares.

Least squares, this measure here, is gonna charge you a lot if your signal goes like this and then goes down. It's gonna – this, you're gonna run up a big bill here. You'll run up a bill here, but the bill here will be much bigger, in the sense of bigger squared. Everybody got this?

So this is total variation. It's been used since the '80s, actually, in image processing, very successfully, by the way. Audio reconstruction. I mean, this – all the old recordings and stuff like that have been, a lot of the wax things have been, recordings have been reconstructed, I mean, almost magically by things like total variation denoising. So I mean, it's actually just amazing.

Okay. So let's just look at an example. Here's a signal. Here's the original signal, which we don't know, by the way. What we get is this. Now, you know, honestly, from a signal processing point of view, this is not a big deal. I mean, you look at this and you say, there's a frequency spread, there's a nice spectral spread between the signal, which is obviously low frequency, and the noise, which high frequency.

Everybody would know what to do. You need a low-pass filter, you need to smooth it. Okay, so – by the way, guess what taken off – guess what, if you regularize with this, guess what you get? You get a low-pass filter. Is it causal? It's a low-pass smoother. It's got an impulse response that goes both ways. It decreases both ways. So that's what it is. You can check. Check. I'm okay.

All right. So, yeah, you can check. It's a nice, smooth kernel like this, it's a convolution. It's almost a convolution, actually, I should say. There's sort of end effects, but okay.

Okay, so here are the smoothed signals. These are smoothed with different – these are three different points on the optimal trade-off curve between smoothness and misfit, okay? Now by the way, if someone were kind enough to tell me what the RMS level of my noise was, this would be easy, because I'd wiggle the parameter until the imputed noise – that's xhat minus X corrupted – in norm or RMS value has about what the noise level is. So if someone were kind enough to tell me the noise level, what you pick on the regularization path would be very, very easy to do.

So okay, so here, you know, and these are kind of obvious that I guess you might say this is too smoothed. This is maybe a smoothed right amount, and that's not enough smoothing here, this one, so these are just three values here.

Okay. Let's see what happens on an example with total variation reconstruction, and it's not – it'll give an idea of it. There's a question?

Student: Yes, on the previous slide, what is X going on the upper left for? I mean, you wouldn't know that, would you?

Instructor (Stephen Boyd):No, you don't know that. If you know this, then that's the – you want to make a good estimate of this, given that. That's all you wanna do.

Student: Okay, so you don't really know that the top upper one on the right is too smooth, because you don't know if it –

Instructor (Stephen Boyd):That's correct.

Student: Okay.

Instructor (Stephen Boyd):Right. You got it. You wouldn’t know. There has to be – I mean, which one of these you pick has to be more. But of course it's true in any multi-criterion problem, right? So you can hardly defend on an optimal risk return curve. You can't say this is better than that, or what risk is better than – you know, what return is worth what risk. That depends on other things. It's the same here.

Okay, let's look at total variation. Total variation works like this. Here's the original signal. By the way, the signal is chosen so the traditional linear signal processing is gonna fail on it, okay? So here's traditional linear – so here, the signal is something that's kind of smoothly varying, and then has, in addition, these jumps, okay?

Here's the one where it's corrupted, and if you do traditional linear signal processing, which is what the taken-off style regularization would be, you can see your choice is something like that. If you wanna get rid of all the wiggles, you're gonna smooth out this sharp transition, something that was basically one sample transition is gonna turn into something that is – I don't know the scale here, but 50 samples.

Why? Because of that square. And actually, a signal, if X moves rapidly, you're gonna get charged a lot for it. You're gonna get charged squared for it. So this is that. Okay, so by the way, in signal processing it would be something like this. If this was an audio signal and you smoothed it – if you low-pass filter too much, then things like the attack on a snare drum or kick drum are muddled now. And I mean, it's total – it's obvious, right? That that's exactly – you can hear what – well, if you know, if you have this background, you can look at that and you know what it's gonna sound like, okay?

And these would just be muddled. So that's no longer a snare drum, that's something more like what I wanted to do to that flag that was flying around.

So, okay, let's look at total variation denoising. Well, total variation denoising does this. Here it is. I mean, these are just different levels. I have a weird feeling these two are switched, don't you? Yeah, for sure these are switched, right? Because that's got more wiggles in it, I think. So I think these two are actually – I should check that. Wanna check it and see if I switched them?

Student:[Inaudible.]

Student: The bottom line is [inaudible].

Student: Because the whole thing is [inaudible].

Instructor (Stephen Boyd):The whole thing is switched. You're right, thank you. Thank you. The whole – they're all – it's upside-down for no good reason. All of my other ones went from too smooth, just about right, under-smoothed. Okay, so these are just – sorry, switch them. So here's the too-smooth version. Actually, now you notice something very interesting here, and this is predicted from L-1. You're actually – one of the objectives, the second objective is the L-1 norm of the first order difference.

If you just apply this intuition that any time you – if you throw in an L-1 norm or something like that and minimize it, with other stuff around, the argument of the L-1 norm is typically gonna be sparse.

If the – what can you say about a signal whose first order difference is sparse? There's a name for a signal like that. No, not constant.

Student: Piecewise.

Instructor (Stephen Boyd):It's a piecewise constant. Constant, that's the name for a signal whose first order difference is zero, okay? So, but the name for a signal whose first order difference is sparse is called a piecewise constant signal. And indeed, as you crank up the parameter on the total variation, you start getting piecewise constant things here. Okay?

Now interestingly, the exact – the sharp transitions are actually preserved. But you'd expect that. That's exactly what you'd expect, because the point is that in L-1, once – you know, if you have to have a big residual, no problem. In L-2, that's not the case. A big residual is a big residual squared. And what's interesting here – I guess this is the middle one – is that something like that is actually pretty good.

I mean, I don't know where the right thing is in here, but this'll work pretty well. So total variation denoising is used in image reconstruction, audio reconstruction, and it works, like, amazingly well. I mean, just – and it actually has been, like, it's actually used in a lot of commercial applications too, so it's used. Actually, in images, I should say something about this just for fun.

In images, what you're really doing is you're looking at something like this, is you would have the total variation – let me just write it – is if you have a function in let's say two variables, X and Y. You'd actually look at something like this. The equivalent of the least squares thing would be something like this. You might have – doesn’t matter, here, zero one squared, okay?

So that would be the equivalent of taking off regularization, and by the way, this would be sometimes called – this would be a Laplacian regularization. So that's what people would call this, okay?

Total variation is this. Okay, by the way, that's still a 2-norm. That's a 2-norm because [inaudible] is like this. So, or if you like that's integral – zero one squared – integral squared plus, partial F, part Y squared. Okay? So it looks like that, okay? So that would be – this is the analogue in two dimensions. Works really well.

You might even find out that it works well sometime in the future, if we finish that problem. So that's – anyway, that's total variation in two dimensions, and so on. Okay.

All right, let's move on to another topic. I'm sub-sampling out of the book, you should actually read everything in the book, of course, and we'll sort of assume – we'll assume that you have, so. Although there's only really one topic so far I think that was important that we just skipped completely, which was theorems of the alternative, and duality for feasibility problems, but that's okay. We'll figure out a way to have you do some of that.

Okay, the next topic is robust approximation, so here you want to minimize norm-X minus B, but the problem is that you don't know A. And there's lots of models for this. One is stochastic, so you could assume A has a probability distribution here. Oh, by the way, same for B. Actually, in a sense we've already assumed B has a probability distribution; that's what this norm is dealing with.

But here, we could assume A has a probability distribution, and for example you might minimize the expected value of the norm, or for example, expected value norm squared, which the second being occasionally tractable, so that would be that.

Another model is worst case. Worst case would say something like this: you'll be given a set of possible values of A, and you would minimize, for example, the maximum of the worst case residual over that set. Okay? That's another version of it. And in fact, you'll actually hear people – amazing to hear actually grown people have empty philosophical arguments about which of these is better, or something like that. It makes no sense.

The answer to which is better is, of course, it depends entirely on the context and application, period. So people who advocate worst case would say things like oh, no one ever really knows the probability distribution, or something like that.

Anyway, the good news actually is that a lot of the methods coincide, and to first order, they're actually often the same. So, okay. All right.

Now by the way, these are both convex problems. Why? Because if for each realization of A – I mean, that's obviously a convex function and an expectation over a random variable, which with probability one is a convex function, it's convex. So these are all convex problems.

However, there may be no particularly good way to write this – you know, no closed form or anything you can actually compute with. There's Monte Carlo methods; they would actually work quite well. And there are methods to deal with both of these.

Same here. So one of the – here, too, it is convex – there's no doubt about it, that the worst case residual is a convex function of X – very complicated. It's just a soup over a family of convex functions. Now the problem is in evaluating the soup here. So if there's a case where you can't evaluate that soup, then this is convex but it really doesn't do you any good, because you can't even evaluate the function.

And there are lots of cases where you can. So let's just look at an example – I mean, a really simple example. So here is – what we're gonna do is this. I mean, it's this dumb, that's what we're gonna do. We're actually gonna have a line segment of As. So you're a matrix, which maps X into Y, is actually unknown and it lies on a line segment parameterized by U here. So it's a line segment. U equals zero maybe is the nominal value, something like that. So if I take – if I just ignore it and I just said A-0, A zero is what happens if U equals zero. That's the nominal value. And I just minimize norm AX minus B. I just do least squares here. That gives me X nominal.

I could then – and this, we can easily work out what this is. It's actually a regularization problem. But X stock minimizes the expected value of the norm squared here. Then – and that's with a uniform distribution on minus one one. And then you can actually work out the worst case one – actually, that's extremely straightforward. In fact, we can even do it right now.

Actually, someone tell me how this was done. How would you minimize that? How do you minimize the – it looks hard, right? Because I'm asking you to minimize the maximum over an infinitely – over a line segment of least squares residuals. Any – somebody gonna help me on that? How would I do it, do you think?

Let me ask you this: for fixed X, what kind of function is that in U? For fixed X. It's what?

Student:[Inaudible.]

Instructor (Stephen Boyd):Oh, it depends where my fingers are – there. Now it's affine. Okay? And now what is it?

Student:[Inaudible.]

Instructor (Stephen Boyd):It's quadratic. It's a quadratic function, okay? Now quadratic functions over intervals, convex quadratic functions, let's talk about that. Where do they achieve their maximum? Boundaries, period. No exceptions. Therefore, this soup is actually the max. it's actually equal to this, right? It's equal to the max of two things: norm – A-0 minus A-1, X minus B squared, and A-0 plus A-1 X minus B squared. Everyone agree with me? That's it.

Now we're back in business. Everybody knows how to solve this. If you're using a system that does compositions for you, like CVX, you'd type this in, literally, okay? If you are not using such a system or you want to know what, in fact, CVX did, that's completely trivial. You introduce a new variable T, and two constraints, that's less than T and that's less than T, and then you go – now you have a quadratically constrained quadratic program. Everybody got this?

I'm not even saying these things, because they're sort of obvious, but I just thought I'd go over this one. Okay.

Let's look at the results. Well, they're hardly surprising, and this is for some random A-1, A-0, and so on. Here's what happens. If you look at the nominal choice of X, it minimized the residual when U equals zero. So it had to be better than the other two. It had to be, by definition, it minimizes the residual, okay?

So in the other two you gave up, this is the one – this is the stochastic one, and it minimizes the average – if you had a uniform distribution on here. So for example the integral of this function over this thing, that's this one, should be least for this middle curve. And I believe that's gonna be the case. Actually, you can see it's the case. Obviously, it beats the nominal one, because for the nominal one, when A moves away from the nominal, you start paying badly, okay?

And then the worst case one is this one. And in fact the worst case, by the way, is equal on the two sides. I mean, it had to be when you minimized this. They're both – it doesn't have to be, but it would typically be these would both be equal, and you can see it's nice and flat, okay?

And so just roughly speaking – and this is a stupid problem with one parameter unknown, but the point is here some people would call this a robustness performance trade-off – something like that. So norm AX minus B is sort of at the nominal value – that's your performance. And you might prefer this worst-case one, because in fact it's giving you a small residual even when A changes, okay?

And so I mean, actually just ideas like this is the second reason why you would regularize things, basically. So, I mean, regularization is a very simple version. This turns, for example – where is it? This one. This one turns into a regularization problem. I won't do it; you could do it easily, okay? I mean, it's as easy as this – even easier, it's an integral, okay?

So that's actually a regularized one. And so the second interpretation of regularization is that you're actually solving a stochastic robust problem, okay? So, yeah?

Student: Is there a reason you don't take an expected value of A or [inaudible] A before [inaudible]?

Instructor (Stephen Boyd):Yeah, we could do that. You could – you tell me. What happens if you look at this? What if you solve that problem? And actually, I'd like you to compare that to this. If you remember. Or you could even do this. It's a sharper one. You know the difference?

Oh, by the way, you see this one? That is what the nominal one is, right? In this case. That's the nominal one. So minimizing this, if U has a uniform distribution here on minus one one? This is exactly simply throwing in the nominal and doing it. So what's the inequality here?

Student:[Inaudible.]

Instructor (Stephen Boyd):[Inaudible] it has the name, now which direction does it go? I never remember this, by the way. What is it?

Student:[Inaudible.]

Instructor (Stephen Boyd):It's like that?

Student: Yeah.

Instructor (Stephen Boyd):Yeah, okay, good, that's it. My rule of thumb is basically wiggling makes things worse. So this is when A wiggles, so this is worse, okay? So yeah, you can think of this – by the way, people would have all sorts of fancy names for this. Some people would call this, this is the "certainty equivalent problem." I mean, that's one name for it. Very fancy name for a very stupid idea, in general, which is just plugging in the – although it's the first thing you would do, right?

I mean, of course you'd start by solving this. You'd know that you're getting an – that the actual performance when things wiggle is gonna be worse by our friend Jensen, okay?

So, okay, let's look at a couple of these and see how these work. The stochastic robust least squares, I mean, this is actually kind of easy. You just take – let's take A is A bar plus U, where U is random, at zero mean, because if it didn't have a zero mean I could plug that into here. So in particular, expected value of A is A bar, but it doesn't matter.

And we'll have an expected value of U transpose U as P, okay? So that's actually all we're gonna need. If we wanna minimize the norm squared of this, if you just – you just work out what it is. It's just algebra. Which I won't even do it, but if you expand this, I mean, that's certain and you just multiply it out, you get this.

There's a cross-term, which is expected value – two times expected value of UX transpose times this, but expected value U is zero, so this cross-term goes away and you get this. The expectation goes inside, and you get that. And now you see a beautiful thing: that if you minimize expected norm squared, where the expectation is over the matrix A here, then it turns into nothing but taken-off regularization. It's quadratic regularization.

So you can actually go back and look at taken-off regularization. If someone asks you what are you doing, you can say well, I'm regularizing. And they'd say why? And you'd say, well, I don't want X big, and so I'm doing a – I've scalarized a bicriterion trade-off here, and delta is a parameter. And if I tune it, if I make delta bigger, my X will be smaller but I'll get a worse fit.

And if I make delta smaller I'll get better fit and bigger X. And if they're not satisfied yet, then you say all right, okay, here's the deal. I actually wanna minimize expected this thing here, and I'm taking into account statistical variation in A. And then that's what this is. And then delta has a very specific meaning, so here, delta is literally – what this – if expected value U transpose U is delta – in this case it's delta I, then in fact we know exactly what delta means.

It tells you about a – it says that the components of A each have variance delta exactly. Did I get that right, or is that the columns? Did delta divide by N or something? No, not [inaudible] because it's delta here. I mean, if I'd thought about it, I would have written it delta squared, but I didn’t think about it, and I think I may even have – is this right? Because U transposed – I think that might even be right.

No, maybe it's just – I think it means this. It means expected value of UIJ squared equals delta. Yeah, I was gonna write the square, but I didn’t. I think this is right – that's what it means. So the regularization parameter therefore should be the variance in the entries of A. So that's a partial answer to – or you could work it either way.

It says whatever taken-off regularization parameter you're using, you can back-interpret it, you can give a statistical interpretation as you are actually protecting yourself against statistical variation in A on this order of magnitude with variance delta. That's the other way to do it.

Okay, now you can also do worst case. By the way, this has been known for a long, long time – I don't know. Although for some reason, the references I've seen, I haven’t seen them go back, like, before the '50s. but this has been obviously known for, like, a hundred years or something. Maybe not quite in this form, but something like that. I'm sure it's known.

Now we get to some stuff that's new, and new means last 10 years, meaning that nobody 15 years ago knew it. So let's do worst case robust least squares. So what we're gonna have is this is an ellipsoid of matrices A. But there's lots of these variations. We have one queued up on the homework for you, so. Lots of variations on this.

We're gonna do robust least squares, except the way it's gonna work – game actually is the correct thing, I was gonna say it – is this. You're judged by the following. In ordinary least squares, you say the [inaudible] norm AX minus B squared. In robustly squared, actually, A lies in an ellipsoid, a known ellipsoid. That's an ellipsoid.

It's the image of the unit ball under a linear affine mapping from some dimension – Euclidean space – into the space of matrices. So that's an ellipsoid. Possibly degenerating – could be flat or something – but it's an ellipsoid. And so I give you an ellipsoid, and you'll be judged, if you – you'll commit to an X and then we'll work out the absolute worst possible A matrix for your X. we'll work out the worst thing.

And we wanna minimize that. Anyone can look at that and say that that's a convex problem because it's a supreme [inaudible] bunch of convex quadratics. But the [inaudible] bunch of convex quadratics is obviously not quadratic, okay?

This can be solved – in fact, there's a whole lot of these problems that can be solved exactly. This is one of them, and the details we won't be able to cover here, but I'll just give you the flavor of how it works.

If you want to – this function, you simply write it – well, by definition it's the soup over U of this thing here, like that. And it's a strange problem, right, because you're maximizing over U and then you wanna minimize over X. So it's a bit complicated, but in fact if you fix X and then you right P of X is just P, if you wanna maximize, PU plus Q norm squared, subject to norm squared less than one – oh, let me just ask you a quick question – is that a convex problem? Where the variable's U? Careful. Is it a convex problem? Careful.

Student: Fixed X?

Instructor (Stephen Boyd):The problem – here, right here – is that convex? The variable is U. Forget X; X is gone. Someone's committed to X already. In fact, you know what this is? We're doing the worst case analysis, that's all. So someone has basically – up here we were trying to do worst case design, forget it. Somebody says here's X, and we wanna find the worst possible residual over an ellipsoid of A matrices. So that's called worst case analysis, and I just wanna solve that problem. Is that a convex problem?

Student: No.

Instructor (Stephen Boyd):No. It looks like one. You're maximizing the norm – norm squared. That's maximizing convex function. Okay. By the way, you might guess you could solve this, because watch this – I need another thing. Actually, if that fly could come back and I could train it to land right – in fact, I'd like it to land right on the Q. Block out the Q, okay? Can you solve that problem? Why?

Student: If [inaudible].

Instructor (Stephen Boyd):Exactly, right? So although – so here's an interesting fact. If I were to get rid of that, that problem, though non-convex – oh, here he comes. Okay, that problem, though non-convex, we can solve. That's trivial. This is solved by – it's the, yeah, find the maximum singular value of input direction and – well, it doesn’t matter, I'm not even gonna go into it, right?

You know how to solve this, this is very easy, even though it's a non-convex problem. So you might not be surprised to know that if you put a Q here, it's still solvable. So it turns out the Lagrange dual of this problem is this – it's an SDP. Okay? This is one of the three or four very famous problems which is a non-convex problem with zero known, zero duality gap. So by solving this, you actually get the same value of that.

Now wait a minute – this is a minimization problem. And now let's reintroduce the fact that P varies with X. We're done. It's just – it's very simple, we just end up with this. Okay? And it's an SDP. So, I mean, I don't expect anyone to get this right now. You can look in the book to sort of – you know, to look at these things.

There's actually – we have an appendix with the three or four most common non-convex problems with zero duality gap. There's actually at least one or two of them is worth knowing. Something called the X procedure, which goes back into the '40s and earlier, and you should actually read that. It's in one of the appendices.

It comes up in computational – it comes up all the time. It's actually rediscovered every 10 years, so. In different fields. Not in each field every 10 years. In different fields, it's rediscovered. So there's huge numbers of papers on it. It's got different names, and I don't even believe it's a connected set right now, in the sense that there's some fields I don't even know about where they have a very powerful lemme or something due to, you know, let's say Johnson or something.

This is probably computer science, because they think everything was invented in the last four years. So surely it's known in computer science, someone figured it out three years ago, and they imagine no one else has ever heard of this before. But in fact it's hit all fields.

Bottom line is this – is you actually end up with an SDP that solves the ellipsoidal worst case least squares problem. And I'll just show what some of the results are. I mean, this is a really stupid one, this is just for a disc of A matrices, okay? And all we did was just took a uniform distribution in Us – and by the way, the problem was to minimize the worst case, not an expected value over a uniform distribution.

But just to give you a flavor of the robustness, and this is sort of what you get. So if you ignore robustness completely, you get something like this. Note that usually, it's horrible. There's a name for this – what's the name for these points over here? What's the name?

Student:I don't know.

Instructor (Stephen Boyd):Any name. Doesn't matter – I can think of many names. Lots of names, what would you call that? Like, how would you describe it to somebody if sort of you did this and this is what happened? What?

Student:[Inaudible.]

Instructor (Stephen Boyd):The tail?

Student:[Inaudible.]

Instructor (Stephen Boyd):No, no, no, no, it's not – but it's because that's the tail, too. No, no, no, no, no. This is called luck, right? Because you completely ignored the variation, and then the variation in A actually drove your residual down, right? So that's what this – this is called luck. Possibly stupid luck, I think would be the right attribute to add in front of it.

If you wiggled – if you played with ticking off regularization, you get a tighter distribution, and we wiggled with it for it to get the tightest one we could, and it was that. Look at that – it's taunting me. Okay. The robustly squares gives you this, right? So – and all this is to show is that actually all of this stuff in solving this SDP, you've actually done something that in fact 15 years ago no one had a clue was possible to do. So that's actually the only point here.

Okay, so we'll quit here.

[End of Audio]

Duration: 78 minutes

ConvexOptimizationI-Lecture11

Instructor (Stephen Boyd):Well, starting on our next application topic, which is statistical estimation. So last time we looked at estimation from a simple fitting point-of-view, approximation in fitting. Now we'll actually look at it from a statistical point of view. Actually it's interesting because you end up kind of with the same stuff. You just get a different viewpoint. So let me give the broad setting is actually just a – well, it's just pure statistics. So here is pure statistics. It's this. You have a probability density that depends on a parameter. So that's pure statistics and I guess our staff department, our contingent, are center front and will keep me honest or at least object when needed.

So in pure statistics there's just parameterized probability distributions and we have a parameter X and your job, you get one or more samples from one of these distributions and you're charge is to say something intelligent about which distribution, which is to say which parameter value, generated the sample. So that's statistics. So a standard technique is maximum likelihood estimations. In maximum likelihood estimation you do the following. You have an observation Y and you look at the density of the – you look at the density at Y or probability distribution if it's a distribution on like – if its got different points on atomic points.

You look at this and you actually choose the parameter value X that maximizes this quantity. That's the same as maximizing the log likelihood. We'll see actually why the log goes in there. The log goes in there for many reasons. One is if you have independent samples that splits into a sum. And the second is that many distributions are actually log concave this time in the parameter. Now, in fact, last time – so far we've looked at distributions that are concave in the argument. This is concave actually in the parameter and those are actually slightly different concepts. So we'll see this in examples. All right.

So the log of the – I'll just call it a density. The log of the density has a function now of the parameter here. Not the argument here. This is called the log likelihood function. In this problem it's often the case, for example, if some of the parameters involve things like variances or covariance matrixes or they're just parameters that are positive or something like that if they represent rates or intensities. They're clearly limited to some set. So you can add to this problem either in explicit constraint, that part that X has to be in subset C, or another way to just affect the same thing is simply to define P sub X of Y to be zero when X is outside the domain of P, at least the parameter argument.

That, of course, makes the log minus infinity and that makes it a very, very poor choice if you're maximizing. So that's the same as just making the constraint implicit. Okay. So these are very often convex optimization problems. By the way, they are sometimes not, if you do machine learning or statistics you'll kind of start to – you'll get a feel for when they're not. Typical examples would be missing – the most obvious and common example is missing data. But we'll see some examples and we'll focus on the cases where it is. So this is often a convex optimization problem. Remember, the variable here is X, which is a parameter in the distribution. Y here actually is a sample. It's an observed sample. Okay.

Let's look at some examples. The first is just linear measurements with IID noise. So here is what you have. You have a bunch of measurements so you have YI is AI transpose X plus VI. Here X is the set of parameters that you want to estimate, but I guess if you're a statistician you would say the following. You would say that the data YI are generated by a distribution. The distribution is parameterized by X. Okay. So that's what it is and the way it's parameterized is this way. In fact, X only affects, as you can see, the mean of the distribution. It doesn't affect, for example, the variance, but that doesn't have to be the case.

So, I mean, the way a normal person would say this is X is a set of parameter values that you want to estimate, but this is fine. The conceptual way to say this is you have a dist – Y is generated from a distribution parameterized by X. Okay. So we'll take the VI's to be IID measurement noises and we'll let the density P of Z. So we're just gonna make this scalar. It's easier to work out what happens in the case when these are vector measurements. It's very easy to do. Okay. So the density of this clearly is simply if you take YI minus AI transpose X and that, of course, has this density and they're independent. So the probability density once if you fix X, this is the parameter it's just simply the product of these because they're independent samples.

So you get this and that's the density. You take the log of that. That's the log likelihood function, and you get this thing here. So this is maximized – you know, I think I'm missing a sign or something here. No, I'm not. Everything's fine. Sorry. So your job you take the log of this. It splits out and you get the sum of the log of these probability densities and your job is to maximize that. So for this to be a convex problem in the general case in X. Remember YI are samples, they're data. X are the variables here in an optimization problem. What you need to happen is you need P in this case, because this appears in P. That's affine in X. You need P to be log concave.

So in this case, because the parameter appears afinely in P, it's good enough for P to be log concave for this to be a convex problem. So let's look at some examples. Here's one. If the noise is Gaussian, so the density is just this. It's nothing but that. Then the log likelihood function looks like this. It's a constant here, has nothing to do with X, and then minus this term. But if you look at that it's a positive constant times nothing but sort of an L2 residual, right? So nothing else. So, in this case, the maximum likelihood estimate is the least squares solution. So now you know if you're doing least squares you're actually doing maximum likelihood estimation with assuming that the noises are Gaussian.

In fact, it doesn't matter what the variances – oh, sorry. All the noises have to have the same variance. If you do maximum likelihood estimation and the noises of the different measurements have different variances what does that translate to?

Student: Weighted.

Instructor (Stephen Boyd):Yes, there's weighted least squares, of course. There's diagonally weighted least squares. So that's what you do. So you can turn it around and if you're doing diagonally weighted least squares the truth is you're doing diagonally weighed least squares if someone asked you. You could say because I trust that measurement more than I trust that one or something like that, but if a statistician friend stops you and say what are you doing? You say I'm doing maximum likelihood estimation with the noises here having different variances. So that's what that means. Okay. That's fine. So now you know what least squares is. Has a beautiful statistical interpretation.

Let's try Laplassian. So in Laplassian noise it looks like this. It's exponential and you actually can say a lot of things about this compared to a Gaussian. The most important by far is that the tails are huge compared to a Gaussian. So E to the minus X squared is a whole lot smaller than E to the minus X when X is big. So this has huge, huge tails. I mean, obviously distributions with huger tails, however, those distributions are not log concave, but nevertheless. So that's the main difference here. You do this and work out what this is, and actually, all you're doing is just this. You're just maximizing the sum of the log of the probability density. You get a constant, that's from this thing. And then over here you get a positive constant time the L1 norm. So that's what you're doing.

So if you do L1 estimation then what you're really doing is you're doing maximum likelihood estimation with Laplassian noise and this, sort of, makes sense actually. This makes sense that this is Y L1 estimation and you know what L1 estimation does. What L1 estimation does is it allows you to have some large out wires. Where L2 estimation would never allow you to do that. In return, it will actually drive a lot of the smaller residuals to zero. For example, to zero or small numbers. And now you can actually justify it statistically.

You can say, well, you're really assuming that the noise, in this case, is more erratic. You're saying that, in fact, V does not fall off like a Gaussian. If something falls off like a Gaussian then being six sigma out is actually a very, very unlikely event. And you will change X considerably to avoid a point where you're actually making the estimate that one of these things is six sigma out. Okay? If this is Laplassian you are much more relaxed about that. It's a rare event, but it is not essentially for all practical purposes impossible event. And you'll actually allow a large out wire there.

So it all, sort of, makes perfect sense statistically. As another example, you have uniform noise. So just suppose the noise is uniform. Which is interesting because it basically says this noise here cannot be bigger than in absolute value of A. Absolutely cannot be. On the other hand, between minus A and A you have absolutely no idea. You're just completely uncommitted as to what its value is. It's not even centered at zero. I mean, it's centered at zero, sorry. But it's not concentrated at zero. It's not more likely to be small than it is to be large. It's completely uniform in there. Sure enough the log likelihood function is this. It's minus infinity unless all of these measurements are consistent, sort of, within A. They have to be.

But then it's just a constant. So here the log likelihood function actually takes on only two values. It's a constant and it's minus infinity. So any point inside this polyhedron is a maximum likelihood estimate. Okay. So that's what happens here. And, of course, it's not unique and so on and so forth and if you were to add to the uniform noise even the slightest tilt towards the center you might get something else or something like that. So what this does is it allows you to translate or to talk statistically about your fitting, your penalty function. So, for example, lets do one.

Suppose I told you – suppose I'm doing fitting and I'm using a penalty function that looks like this. Okay. So it means I don't really – if your residual is within plus minus one I don't care at all. I just – it's fine. I'm not even interested. Once it's between an absolute value less than one it's fine. Then it grows linearly and so I want to know what's the statistical interpretation. This is maximum likelihood estimation. What noise density? What's the imputed noise density here? What would it be? You do. This is maximum likelihood estimation so what – this corresponds exactly to maximum likelihood estimation. What is it?

Student: It's uniformally [inaudible] and the [inaudible].

Instructor (Stephen Boyd):Exactly it. Yeah. So if you're doing dead zone linear penalty, which you could do and just defend it on very simple grounds and say, well, look if the residuals – look, I can't even measure it that accurately. So if it's between plus minus one I don't even care. That's fine. That's as good as my measurements are anyway. It makes no sense. And then you say, well, what about linear? Why not quadratic out here? Linear would say something like, well, sometimes there are some pretty large residuals or pretty large errors and things, so that's why I have this linear and not quadratic. Okay? So that would be it.

The statistical interpretation is exactly this. You're doing maximum likelihood estimation of Y equals AX plus V. The VI's are IID and they have a distribution that looks like this. That's supposed to be flat by the way. I don't – there. Okay. So it's a uniform distribution between plus minus one and then it falls off. It has exponential tails outside. So that's what you're doing statistically. Makes perfect sense. Here you'd adjust everything to have integral one, of course, but that makes no difference, in fact. It's just that that's the shape of what you have. So if you do uniform – by the way, this is, of course, log concave because the negative log of this is that. And, in fact, that's how I got – I mean, that's how this came by flipping this and then taking the X. Okay. So that's how that works.

So that's all you do. Okay. So there's more on that, but that's something you can do in homework and reading and stuff like that to sort of see how that works. And this is very simple case. You can actually get to the case where you're estimating things like covariance matrixes and things like that or other parameters and it's actually more interesting, but that's the basic idea. Okay. Let's look at another example of something different where the noises are not additive at all. They enter in a complex non-linear way. So we'll look at logistic regression as sort of a canonical example of other classes of maximum likelihood estimation problems that are convex. So let's look at that.

I have a random variable that's in zero one and its distribution is the following. It has a logistic distribution so the probability that Y is one is E to the A transpose U plus V. It's E to a number divided by one plus E to the number. So that looks like this, I guess, if you plot this function like that, so that's zero. And what this says is – actually let's even take a look at this. If this – for now just treat this as a number. A transpose U plus V. If this number is let's say zero it means that sort of the equiprobable point. It says that the probability that Y is one is half. If A transpose U plus V is like two or three, then this is very likely, but not perfectly certain to be one.

If this number is say minus two or three this is also – it's very likely to be zero, but not quite. So the transition zone is where this number is, let's say, between plus minus one. I mean, you can chose that number some other way. But roughly speaking when this number is zero that's the equiprobable point. When this number is between, I don't know, plus minus a half, plus minus one, you can throw in your favorite number in there, the probability is some sort of number like 10 percent, 90, between something like that. You can work it all out. When this number is things like three and four, this is overwhelmingly probable to be one here and if it's negative three or four it's overwhelmingly probable to be zero and that's, sort of, a picture like this. Okay.

Now, I guess to the statistician here A and B are parameters. So those are the parameters that generated the distribution and so your, our job is gonna be estimate A and B. Now, you actually – you can't estimate A and B if I just give you one sample from this. So you're gonna be given multiple samples. In other words, I'll give you a bunch of samples and for each sample I'm gonna give you U. So U here is – these are often called explanatory variables. So they would be other things that you measure and associate with this sample.

So this would be, for example, I don't know. If you were admitted to a hospital this would be things like blood pressure, weight, blood gas concentrations, all things like this. And this could maybe be if you died or something like that, right? So that would be that so – yeah, hopefully things will end up over here. All right. So that's it. All right. So, okay. So that's what this is. so these are the explanatory variables and what you want to estimate are the parameters A and B to parameterize this distribution. Okay. So what we'll do is we're given a bunch of samples and a sample looks like this. It's a pair UI and YI here. These are people who checked in and this is what happened to them. I guess in this case, zero being the good outcome. Well, we have to decide that we'll make zero the good outcome.

So you get a past data like from last year. You get this data, get a couple hundred of these or something like that and, actually, let's even just talk about what you do with this. We're actually gonna fit A and B. So we're gonna do maximum likelihood estimation of the parameters A and B. By the way, once we have them we can do interesting things because someone can arrive tomorrow at the hospital, we can evaluate their U, evaluate this, and for example, if this turns out to be plus three that's not good for them. If it's minus three I guess we can reassure them or something like that. Okay. All right.

So this is the idea. Now, how do you do this? Do you want to work out the log likelihood as such is nothing, but the likelihood function is the product of these things because we're assuming these are independent samples from a logistic distribution. So you take the product of these. You could take, which is this, and what we've done is we've simply reordered the samples so that the first K1's were the ones, I guess, who died in this case and the last M minus K plus one or something like that, whatever this is, are the ones who didn't. So that's – you just reordered them. And, in that case, you get the – you just multiply all this out. You can see that in the denominator you always get this thing. That's for all of them.

And the numerator take this and you take the log of this and this thing is the log of the product of the X. That comes out here as this afinely term and this one over here is minus log sum X. Now, of course, log – well, by the way, why did I call this log sum X? Well, I'm going fast now, but someone want to justify? How did I know just immediately that this thing was convex?

Student:A longer sum of the two.

Instructor (Stephen Boyd):Yeah. But there's a one there. That one is expo of zero. This is the log of E to the zero plus E to the A transpose UI plus V, like that. Okay. So it's log sub X. It's the two-term log sub X culled with zero comma A transpose UI plus V. Okay. So this is convex in this argument that's afine this whole thing is convex. With a minus sign it's concave. So this is concave. Okay. So that's the picture. By the way, if you want you can actually draw this function as a function of AI transpose U plus – yeah, that would be called the logistic. Well, if I put in a negative sign here, which I haven't done yet, it would be called the logistic loss function. And it would be something like this. I wonder if I can get it right. Let's see. Which do you want? If this is small, then that is about zero and it's that, so I think it looks like that at zero.

I may have gotten that right, but if it doesn't look like this then flip it over and if that doesn't work flip it this way and if that doesn't work flip it this way again. So in one of those orientations with some number of minus signs and so on the logist – I guess actually people do logistic loss they refer to it as this. It's a function that looks like that. That's where this gets linear in this case. Approximately linear. Anyway, so but this is something to be maximized. But is this covered in – I know a bunch have taken 229, ES229. Is this there? Come on. Is logistic regression covered in?

Student: It is, yeah. Yes.

Instructor (Stephen Boyd):It is. Okay. All right. So here's just an example. It's a super simple example. This is, as with most examples that you can show on a slide, they're examples of what – they're examples where you definitely did not need any advance theory to figure out how these things work. Okay. So if, in fact, there's only one explanatory variable, which is here, U, and then a bunch of outcomes here, then you definitely don't need anything in advance. Just look at the data with your eye. So that's not the point of this. I'll give you some examples of where logistic regression works. Where it's – we're not talking what you do with your eye.

So here's a whole bunch of data. It's 50 measurements. And each measurement consists of this. It's a value of U here and then it's a one or zero. So that's all it is. I mean, this could be anything. All right. And you can see roughly the following. That when U is larger you – there you get more ones. I mean, you can see that because there's sort of a concentration up here and fewer down here and you can see that if U is smaller there's fewer – you're more likely to be zero. Okay. But you can see there's lots of exceptions. Here's somebody with a very low U that turned out to be one and somebody with a high U that turned out t be zero. Okay.

Oh, I should mention one thing. What do you – well, okay. What this shows is A and B have been estimated. A and B do nothing but control what here in the logistic distribution? B controls the point where this is neutral and A controls the spread. It's the width of the transition zone. So A controls how stretched out this thing is and B controls, sort of, where it is and you can see that it is lined up about at what would be a very good decision point if you were forced to make a hard classifier here. It's lined up quite well with something that approximately separates, not perfectly, but approximately separates these cases from these.

And this spread here tells you roughly how much indecision there is. Okay. Let me ask you a couple of questions here. What do you imagine would happen if the data had looked like this? What if there were none there and there were none there? Okay. So basically there's this data and there's this data. What do you think? What do you think the logistic – well, first of all, let's just talk informally here about what you think the logistic estimate – the logistic maximum likelihood is gonna be? Just draw a picture. What do you think? What's it gonna look like? I mean, like this? No. It's gonna be much sharper. In fact, it's gonna be infinitely sharp. It's gonna basically look like this.

It's gonna be a perfect thing like that. Right? And the truth is, this thing can actually move left or right and it makes absolutely no difference. And the point is that actually the data here is actually linearly separable, right? There's a perfect classifier. That corresponds exactly to the logistic regression problem being unbounded. If you're doing maximization it's unbounded above. Okay. So unboundedness above of the log likelihood. It means you get an estimate. It basically means you can make the the log likelihood function as large as you want. Okay. And that corresponds to this perfect separation like this. Okay.

You have to check me on this. So this is the – okay. What would be uses for this? I am not quite sure, but I believe actually some of the best spam filters are done using logistic regression. You hear both things. Something called support vector machine, which you'll see very shortly, and logistic regression. So my guess, I think it's both. So that's how that works. There, of course, the dimensions are very different. They're not one, right? You've got thousands and thousands of features and things like that and your estimating over very, very large data sets. Okay. This makes sense?

So this is an example where it's not linear remotely obviously. I mean, this here it enters in a very, very complicated way. The measure – if you like to think of this is a measurement or an outcome work this way. By the way, I should mention to those of you in areas like signal processing and things like that you might think, well, this doesn't have anything to do with me. Actually, I beg to differ. This is a one-bit measurement. Okay. So this is exactly – well, with a particular distribution. You change the distribution it's something else. So you will – if you do signal processing with low bit rate measurements, including, for example, single bit measurements, you will exactly get problems like this. Not quite this one. If it's Gaussian and then you take a sign you're actually gonna get – it's called probit regression and you get a different function here, but it's the same everything otherwise.

It's kind of the same. Okay. So that was our whirlwind tour of a more complex parameter estimation – a maximum likelihood estimation problem. Okay. And let me mention, actually, with this one I can actually mention some of the cases where the canonical cases where you get non-convex problems and this you probably look at in machine learning and other applied classes and things like that. What would happen is you get a bunch of data samples like this, but for some of the samples some components of the UI's would be missing. That's the missing data problem. And there it's not – the resulting problem is not convex. Okay.

There's some very good uristics that you can imagine. Like assuming a value of U, carrying out the regression, then going back, plugging in the most likely value, you know, estimating U and alternating between these two and these are all schemes that are used and so on, but, okay. Now we're gonna look at hypothesis testing. So how does hypothesis testing works? It works like this and for this we go back to the simplest possible case. You can get very complicated other cases like multiple hypotheses and not just one, but multiple ones. Sorry, two, but, sorry. The single hypothesis testing is actually quite simple, but anyway. So we'll go to the simplest one.

This is where you have two and we'll just take a random variable on the letters one through m. So just very, very simple random variable. And what's gonna happen is either a sample, which is literally – it's just an integer between one and m. It was either generated by the distribution of P or the distribution of Q. So these are non-negative numbers that add up to one. Good as vectors actually. That P and Q. The distribution because we have a finite set here. So that's the question. I give you a sample and you – I give you a sample or I give you a couple samples of something like that and your job is to estimate was it this distribution or this one?

Now, of course, this could be very easy. For example, if X turns out to be one and this is kind of intuitive, and P1 is .9 and Q1 is .002, right? Then it's just intuitively clear. It's quite clear. It's a very good guess. By the way, not a perfect guess, but a very good guess that, in fact, X came from this distribution of P and not Q. Okay. So that's roughly – it's not that unobvious what to do here as in these other things. It's not that it's intuitively unobvious. The question is how exactly do you estimate and how do you make a better estimator than an intuitive one. So we'll look at the idea of a randomized detector. So if a randomized detector looks like this.

It's a two by n matrix. So it looks like this. And what it means is this. Each column is actually a probability distribution on the two outcomes like hypothesis one and hypothesis two. Okay. And it says that if this is the outcome that occurs you should guess one with this probability or guess two with this probability. So that's the idea. Now, of course, often these things look like that. Okay. Something like that. What this is if this is what you observe in X you simply guess it came from one. If this is what actually is observed you guess it came from hypothesis two or Q. Okay. So if these are zero one entries it's a deterministic detector.

And the deterministic detector is silly. It just means basically you have partition X, the outcome space, into two groups where if the outcome is in one set you declare that you guess its hypothesis one. If it's in the compliment you say it's hypothesis two. I mean, that, sort of, makes sense. However, you can have something like this. You can have something weird like that. That means that if this is the outcome of observed you then go off and you toss a coin and with 80 percent probability you guess that it was hypothesis one and with 20 percent probability you guess it's hypothesis two. Now, by the way, this is sort of like randomized algorithms in computer science.

This just looks weird like why on earth if you're trying to make an intelligent statement or guess what happened it doesn't seem like gambling on your own is actually gonna help anything. I mean, why on earth would you say, well, tell me happened. You go, excuse me for a minute and get out a coin and flip it. It just doesn't look – there's something that doesn't make sense. Which is kind of like in a randomized algorithm, right? When someone says, here's my problem instance. I'm talking computer science and you say how do you solve it? And you go hang on just a minute and you get out the coin and you start flipping it and you say, look, what are you doing? And you say, no, I'm, gonna try to solve your problem. You say my problem has no probability in it.

I gave you a problem instance, please give me the answer. You go hey, back off. Come on. And you keep flipping the coin. It just looks – you know what I'm saying. After you get used to this intellectually it's okay, but at first it looks very odd. So the idea of having a deterministic detector – also, it's weird. Because you say, hey, output three happened and you go, yeah, sure that came from hypothesis one. Then the next day you say output three happened and you go, yep, that was hypothesis two. All right. So this looks very inconsistent and strange. Actually, we're gonna soon see what a randomized detector can do for you. I mean, it's still weird, I guess. It's like if you're in physics you have to – some day you have to either come to some – you have to sum an equilibrium, for example, with quantum mechanics and things like that. It doesn't make any sense and same with these. Or not. But then you just know it's an unresolved intellectual issue. Okay.

So this is what a detector matrix is and it describes a randomized detector and if you're uncomfortable with that you can make it all zero one matrix. Of course, there is a one in each column and this becomes an encoding of a partition of the outcome space. Okay. Now, if I simply multiply this matrix T by P and Q. So if I take T and then I multiply by PQ, like this, that's lined up like this, then I get four probabilities here and they're actually quite interesting. So I'll tell you what they are. And I guess we should start with – well, I don't know. We could start with the one one entry.

The one one entry is the probability that you guess hypothesis one is correct when, in fact, the sample is generated from hypothesis one. So this entry is a good entry – sorry, it's an entry that you want near one. Actually, if it were one it would mean that you would be absolutely infallible. You could never – this entry. Let's go down and look at this one. This entry is that's the probability that you guess hypothesis two when X is generated by distribution one. Okay.

So that's the – this is a false positive. We're taking, I guess, hypothesis two to be positive. Okay. So that's a false positive. Now, we'll get to that. This one is two two entry so these two add up to one. I mean, they're like a conditional distribution in fact. I mean, they're conditional probabilities. This entry is the probability that you were – that, in fact, the distribution was generated by distribution two and you guessed correctly distribution two. So that's this one. Again, that's something you want one. And this is – so these are the ones you want small are these off-diagonals that's probability of false positive, probability of false negative and you want those small. So, in fact, what you really have here is a bi-criterion problem because what you really like – well, of course, you'd really like this matrix to be the identity matrix.

That means you're absolutely infallible. That means whenever distribution – it comes from distribution one you correctly estimate the distribution one and so on. You make no mistakes of any kind. No false positives. No false negatives. But you really have – in general, you have a tradeoff between these two false alarm probabilities. Okay. And I guess there's lots of names for this tradeoff. I actually don't know what it's called in statistics. I know what it's called in signal processing. It's called the ROC, which is the receiver operating characteristic, but which goes back to like World War II or something like that. What's it called in statistics?

Student:[Inaudible]

Instructor (Stephen Boyd):Really. Maybe it came from you guys. Who knows. It doesn't sound like it to me, but – okay. All right. So this is – that's the bottom you want and let – we can actually talk about some of the extreme points. Let's see. How small could I make the probability of false positive be and how might I do that? How small can I make the false probability – the probability of a false positive?

Student: Zero.

Instructor (Stephen Boyd):I can make it zero. How?

Student: Always guess one.

Instructor (Stephen Boyd):Yeah. You just guess one no matter what happens. Then, in that case, you will never be accused of having guessed two when, in fact, one generated the sample. Okay. So that – by the way, that scheme also has another huge advantage, which is great simplicity. You don't have to even listen to what happened to guess where it came from. And, obviously, I could make false negative zero as well by simply always saying it's positive. All you do is you always guess it's – sorry, guess it's negative always. And I think we've got it wrong. This was – sorry, this was. To make a – no, sorry, that's right. You guess one here, you guess two always. Okay.

So will – I mean, this is actually correctly viewed as a bi-criterion problem. Okay. So it's a bi – oh, and by the way, what happens if there's multiple hypotheses like 12? What happens if there's 12? Let's talk about it. Suppose there's 12 because this is not exactly complicated stuff. We're just multiplying the matrix out. What happens in the case of 12 hypotheses? Wow, this would be a good homework problem or something like that. So what happens?

Student:[Inaudible]

Instructor (Stephen Boyd):You have a 12 by 12 matrix. And how many of these bad guys off the diagonal do you have?

Student:144 minus 12, so 132.

Instructor (Stephen Boyd):Yeah. But some of them aren't – oh, okay. That's fine. Yeah. That's fine. Okay. Yeah. So roughly 70 or something like that, right? Okay. So we've got a bunch of them, right? And so now you have a big tradeoff where you might have strong concerns about – you might have different concerns about guessing hypothesis three when, in fact, the truth came from whatever hypothesis two. That might be one you really care about making low or something and you can imagine after then it would get interesting, but we're just looking at it in the simple case. Okay.

So you end up with this. If it's a bi-criterion problem. You want to minimize these two things subject to – this just says the column sums in T are one, obviously, and this says that they're non-negative and so this is obvious. By the way, notice that everything here is linear. So if this a bi-criterion LP. Actually, it's a trivial bi-criterion LP. So let's scalorize it, so we put in a weight lambda. If you put in a weight lambda you add this up and this is silly. That's a linear function of T of the entries in T. It's extremely straightforward. This says subject to that. It's not only a little – I mean, so this is one of those linear programs, I guess, that one of those trivial linear programs that you can just directly solve.

It's kind of obvious. In fact, you can optimize each column of T completely separately because that's linear and so it's separable in T. Gives you a sum of contributions from each column. The constraints on T are simply that each column is in the unit simplex and I think you just did a problem on that or you did one a while ago or something. Anyway, and its' extremely simple how to choose? You simply choose this. If P is bigger than lambda Q you choose the first hypothesis. The other way around, you choose this one. So you end up with a deterministic detector and this is called a likelihood ratio. Likelihood ratio test because what you really do is you look at this and you compare it to a threshold and then you guess which distribution it came from.

By the way, this extends to the idea of continuous distributions and things like that and it's very old. It's from easily the 20's or something like that. Roughly, yeah, 20's I'd say. Maybe even earlier, right? This is –

Student:I don't know. I think that's about right.

Instructor (Stephen Boyd):20's? Okay. We'll just say the 20's, right? It goes back to – so now you see you have a nice mathematical way to go back to the simple case. You just – you look. How do you guess which one it is? Oh, by the way, if lambda is one then you simply choose whichever one is the more likely. By the way, we'll see what the lambda equals one means. Lambda equals one – well, you can tell me. What does lambda mean here? What is lambda equals one? Because a maximum likelihood test would simply choose whichever had the maximum likelihood. That corresponds exactly to lambda equals one. What's the meaning in the bi-criterion problem for lambda equals one? Has a very specific meaning. What does it minimize? That minimizes something for false positives and false negative maximum likelihood. What is it?

Student:[Inaudible] of error.

Instructor (Stephen Boyd):The sum? Yeah. Which is otherwise known as the – if you put a lambda – if lambda's one here it means, actually, that you're minimizing the sum of this and this. But the sum of this and this has an interpretation. It's called being wrong. Right? So you're minimi – it's called an error. So you're minimizing the probability of error. That corresponds exactly to lambda equals one. Okay. By the way, this ties back to maximum. Like this is, sort of, the justification for maximum likelihood. So you can say maximum likelihood detector you would then argue, you'd argue and you'd be correct, minimizes the probability of being wrong and you're absolutely neutral. False positive has the same weight as false negative. You have a question?

Student: Yeah. For this do we have to assume that there's kind of an equal likelihood of being distribution E and distribution 2?

Instructor (Stephen Boyd):Nope.

Student: Because it's –

Instructor (Stephen Boyd):We're doing statistics.

Student: Okay. But, I mean –

Instructor (Stephen Boyd):You're not even allowed to say that in statistics.

Student: Okay.

Instructor (Stephen Boyd):I mean, if there are –

Student: If you're Basian.

Instructor (Stephen Boyd):Yeah. If you're – no, no. If you're Basian, no, no, no. We're not doing that right now. So that's – if you say that in front of statisticians be careful. It could be physically dangerous. If there are some Basians around in a room with you and they're large you're safe. Go ahead and say that, but watch out.

Student: Okay. So, I guess, it seems then that if we get a data form and there's a – and we look at P and Q and there's a very small probability that that data point came from distribution P. A very large probability that it came from Q, but if we know that 90 percent of the time –

Instructor (Stephen Boyd):By the way, if you're all ready using all ready, you've identified yourself as a Basian and you'd be all ready in big trouble.

Student: Okay. Well, I don't –

Instructor (Stephen Boyd):That's okay. I'll tell you in a minute. Okay. You're safe here.

Student: Okay.

Instructor (Stephen Boyd):Don't try this out in certain parts of Sequoia. Okay. Don't try it. But go on.

Student: Okay. But, anyway, if you then – if you know that 90 percent of the time it's distribution P then even though it's a small probability that your outcome came from distribution P, the fact that distribution P is much more likely should probably influence your choice.

Instructor (Stephen Boyd):Absolutely. So, actually, this is a very good point to say this. So we're doing statistics. In statistics there is no prior distribution. That you can't say even something like this is more likely or what's the likelihood of it being this or that? You're just neutral. Okay. I'm – by the way, I'm neutral on this. I'm not partisan in any way on this. We have to be careful because a lot of people are in machine learning in this room too and I don't know where their sympathies lie. A peer statistician would never – you're not even allowed to say something like what's – this is a more probable out – which one is more probable? The minute you say that now you're doing Basian estimation.

Now, I should say this. That if you do Basian estimation, then instead of the maximum likelihood you do something called MAP, which is Maximum A Posteriori Probability.

Okay. And it's, of course, quite sensible if you have some ideas about what – which of these things. If I told you ahead of time – by the way, you could redo all of this and it's very, very easy in that case. If you do things like saying, well, yeah, it could come from these two distributions, but, in fact, with 30 percent chance it comes from P and 70 from Q. And you could work out everything that would happen here and you'd want something called the max. Then you could talk about the probability of the actual probability of being wrong. Things like that.

Now, the good news is this. By the way, we could have done that earlier as well in maximum likelihood estimation, like, for example, here we could have done MAP. All that happens is very cool. You add an extra term, so what is a log likelihood function or a likelihood function – a log likelihood function turns into a log of a conditional density. Then you multiply by the log of a prior density and you get regularization of terms here. So that's what would happen. But you need to know just socially speaking that watch out you're treading on very dangerous – just saying things like that can get you in big trouble. What's that?

Student: That wasn't a good thing to start.

Instructor (Stephen Boyd):No, I'm neutral. I'm totally neutral, but just you can say that in the wrong place and be very sorry. So the good news is from the convex optimization point of view they all lead to – so the way that would happen is if I was being – if I had some Basians coming on one direction and I had Daphne Collar coming on one side and who would be the most extreme one in statistics? Who would – Who?

Student: No, I –

Instructor (Stephen Boyd):Whoever you're not gonna name any names.

Student: People are pretty flexible now.

Instructor (Stephen Boyd):They're pretty flexible, yeah. So that was the right answer. And I had a statistical fundamentalist coming in. A group of them coming in on the left. I would – for us it's very simple. You just add a term here, which we would just say is this regularization? It's regular. And they'd say that's not regularization, that's log of a prior. And I'd say, no, no, this is just regularized maximum likelihood or whatever of that. Thanks for bringing that up because I just wanted to – I mean, if actually I think if you read the book it's neutral and, I believe, honest about what it says. It just says if you choose to believe that you're getting a sample from a parameterized distribution and your job is to estimate the distribution that's a statistician. That's statistics. You can do that.

Student: Are also statisticians.

Instructor (Stephen Boyd):Now you know who I hang out with and what they are. Okay, sorry.

Student: It's Frequentists is the –

Instructor (Stephen Boyd):Frequentists, Okay. So I'm not totally up on the details of these very schisms, but I do know that there's been a lot of wars about it and a lot of bloodshed actually over these issues, but for us they're all just convex problems and they're just innocent little extra terms that end up in here. So that was a very long answer to your question. But it's a good – just as a matter of safety to know. Asking that innocent question in the wrong place could get you in deep trouble. So not really, but – actually, maybe it could. Well, we won't go in. All right. So let's look at some examples now. Oh, we have our interpretation of maximum likelihood.

Maximum likelihood says you scan – if outcome three has occurred you scan the two probability distributions, you go to outcome three, and you see which has the higher likelihood. That's a maximum likelihood detector that corresponds exactly to lambda equals one. Lambda equals one corresponds precisely to saying that you treat false positives and false negatives equally. Okay. That's maximum likelihood. That will minimize the sum of the two, which is something like the probability of it being wrong. Okay. Now, you could also do a mini max detector.

You could say, well, I want to minimize the maximum of the two error probabilities. Now, this one actually – of course, it translates to an LP immediately, but this one actually, generally speaking, in fact, in the finite case essentially always has a non-deterministic solution and so let's see what that is. So here's an example where these two distributions and it's kind of look, anybody can figure out what happened here. If it's outcome one you should probably guess it came from P and if it's outcome three you should probably guess it came from Q. Everyone's gonna agree on that. Okay.

And there's actually not much else here to do, right? Because two other outcomes and obviously if it's outcome two you should kind of lean towards saying it was two, but you should equivocate or something like that. Something like that. Okay. All right. So here's the ROC, or Receiver Operating Characteristic, although it's generally drawn this way and I don't know why, but anyway, it often is. So here are the two. There's a probability of false positive and false negative. We've all ready talked about these things. This one is the detector that simply says – sorry, yeah. You have zero false negative that means you just simply always say it's positive. You just announce it's positive and you can have zero probability of false negative.

Here's a tradeoff curve here and actually it's on, of course, it's piecewise linear. I mean that's obvious because you're minimizing a piecewise linear function over some linear inequality constraints. So this curve it's – well, this region is polyhedral and the vertices here correspond to different things. In fact, the vertices correspond to the three thresholds in lambda. Lambda is the slope here and so as you vary lambda to get a tradeoff curve it's kind of boring because if lambda is smaller than this you get that point. As you increase lambda you will click over to three, this point, and it's just some point. These are the deterministic detectors, one, two, three and four. As you increase lambda like this it's always the same.

Right at this point you switch over to a different threshold. Then you keep going like this and then eventually you get so high that the safest thing to do is simply to guess that nothing ever happened and that way you will have zero false positive always. Okay. So there's not many points on the tradeoff curve, but the mini max one that's very simple. It's the maximum of the two and the level curves of the maximum – this dash line here is equal, is the line of equal false probability and false negatives and you can see very clearly that it's not a determinant. It's right in between these two and it's not a deterministic detector.

So if you want to have a detector here which minimizes the maximum probability of making either a false positive or a false negative then you're gonna have to use a non-deterministic – a randomized detector. Okay. Now, I mean, this is all kind of stupid looking in cases like this. That's fine. But if you want to make this non-trivial, it's very, very simple. You just imagine something with 12 outcomes and vectors, which are – and probability solutions on a thousand points. Okay. Very, very simple. And now you want to decide on a detector and now you start throwing in insane things like how much you care about guessing it's No. 7 when, in fact, the truth was No. 3 and you start throwing in how much you care about all these things and all of a sudden it's not obvious exactly how to do this.

Then when you solve an LP you get something that – or an LP or whatever. Actually, it's always going to be an LP. When you solve an LP you'll actually get a detector that will beat, soundly beat, according to that measure whatever your criterion is. Okay. So that's the picture. Okay. So that's that. There's a lot more you can do with mini max detector – with detectors here. And there's a couple more topics in the book, which you will read about. Okay. Now, we're gonna look at our last topic in this whirlwind tours experiment design. This is quite useful, very useful, and I think not that well – in some fields it's not that well diffused, knowledge of experiment design.

Even in areas where people actually often do experiments and get – construct models from data and things like that. Okay. So experiment design goes like this. And we'll talk about how it works. We'll do the simplest case, which is linear measurements like this and we'll just say that the noises are IID and zero one. You can change all of this, but let's just do that. Well, the maximum likelihood of least squares estimate is just this. And the error is zero mean. That's X hap minus X is zero mean and has a covariance matrix, which is this thing. So that's the covariance.

By the way, here the noises all have noise power one. So the norm of A is something like a signal to noise ratio and now I'm talking if you're in signal processing or communications. So that's because I just made the noises have power one. So if A is large – if the norm of A is large that's a very good measurement. It's a high quality measurement. It's a high signal noise ratio. If the norm of A is small – the norm of A is zero, that's an utterly useless measurement because it's just a sample from noise and it has nothing to do with X. Okay. So that's if you want to get a rough idea. A larger A is larger signal to noise. That's what norm of A is. It's literally the signal to noise, something like that ratio because the noise power is one. Okay.

All right. And I'm sort of assuming there norm of X is on the order of one, but that – with that assumption that just scales the signal to noise. It's still the case that if you double A it's, sort of, twice as good from a noise point-of-view, standard deviation anyway as if you hadn't doubled A. Okay. Now, this is the error covariance and it's a sum of diags inverse. That's very interesting. Okay. So we can say a lot of interesting things about this, but first is if that matrix were singular that would not be good. Okay. And what that means is the matrix is singular it means you basically haven't taken a set of linearly independent – you haven't taken enough measurements basically. Sorry, measurements that span X. Okay. So that would be the case there.

So if you've taken enough measurements so that the measurement vectors span our end then that means this thing is not singular and then you take the inverse and, of course, that's the error covariance and now it's clear now I'm gonna be very rough here. To make this matrix – you want this matrix small and we'll talk about what small can mean. It can mean lots of things. You want the matrix small. Roughly speaking the first thing you want is you want what's inside it since that's an inverse to be big. Good. That corresponds perfectly because if A is big – if the A's are big then that means you have high signal to noise ratio and that makes this error covariance small.

Now, the interesting thing is it's not scalar, it's actually matrix inverse and that's very interesting because it means what the error covariance is is not simply – it doesn't depend just on the individual A's. It's actually how they all go together. Okay. Oh, there's one exception. If the AI's are mutually orthogonal then this inverse you can just do it, change coordinates, and do it and they're independent and then they don't interact in any way whatsoever. But, in general, what this says is the following is you take a bunch of measurements, those are characterized by A1, A2, A3, and so on, and then you calculate the error covariance and the error covariance kind of depends on actually the geometry of the whole set of these things. I mean, this is kind of obvious, but that's the idea.

And, for example, if you want to make an error code – a confidence ellipsoid it would depend on this covariance thing. Okay. So experiment design is this. It says I give you a pallet of possible experiments you can carry out. So we'll call those V1 through VP and these are just experiments you can carry out and now your job is to choose some number of experiments that will make this error covariance matrix as small as possible. So another way of saying that is I give you a pallet of possible experiments you can choose and the simplest thing would be I give you 50 possible measurements and I say you may choose 1,000 measurements from these 50. Okay. And you choose a thousand and you want that choice of a thousand from those 50 to be mutually maximally informative because what's gonna happen is you're gonna take all those measurements together. You're gonna blend them, do least squares, and do all the good blending that least squares is gonna do.

Together they're gonna give you an error covariance like that. Let's talk about some choices. If someone said, well, since your one has the highest signal to noise ratio, so I'm gonna choose all 1,000 measurements to have the signal to noise ratio ten because all these others have signal to noise ratio one. Any comments on that? Is it a good choice? If

V1 has a signal noise ratio of ten and all the others have signal noise ratio one then you can choose any of them. So you can – why would you not always choose the first measurement. It's ten times cleaner than any of the other measurements. What's the problem?

Student: This is not an [inaudible].

Instructor (Stephen Boyd):Yeah. You do unbelievably badly here because if A – if all the AI's are equal to V1 this one is ranked one. And it is by a long shot not invertible. Everybody see that? So the point is you're gonna be forced. You can't do a greedy thing and just choose high quality measurements. It's actually something where you have to choose all of them together. Does this make sense? And actually if you're confused it's probably because this is actually quite trivial and I'm just making it sound complicated, but, anyway, okay. That's – but I do want to point this out. Okay.

So you can write this this way. It's a vector optimization problem over the positive semi-definite cone and here's what it says. So obviously all the – it doesn't matter what order you choose the measurements in that's clear because all you're gonna do is sum these dyads here. You just sum the dyads. So it doesn't matter which measurement I do first. What matters if I have a thousand measurements to take I have a pallet of 50 choices. What matters is how many measurements of type one do I take? How many of type two and so on. And we're gonna call those MK. So MK is the number of measurements of choice K I make. And so I get this problem here. Now the MP's are integers and they have to be non-negative and, yeah, they add up to my budget. By the way, this is just the simplest version. I can also put a cost on each measurement.

I cannot only put a pallet of measurements in front of you. I can also put a price tag on every measurement and you can have a budget in money or time and you'd get a – then this budget would be something different. I can add other constraints to this if I want. I can add a time, money, all sorts of other constraints. This is just a simplest case where all the measurements are equal. You have a – you're allowed to choose [inaudible]. Okay. Now, this problem here – if you just say experiment design this is what experiment design is. It's this problem right here. Okay. And this problem in general is hard to solve, but there are some regimes where it's relatively easy to solve.

Actually, the feasible set is essentially the set of partitions of M. By the way, if M is really small, like three, then this is pretty easy to solve, or four or something like that. Then this is easy to solve. The other extreme is when M is very large because what you do is you rewrite this this way and you let lambda K be MK over M and this is now the fraction of the experiments of the M total budget you're gonna use of which you will take experiment type – measurement type K. So this fraction. Okay. Now, I haven't changed anything here. Actually the truth would be something like this. The real problem would have M lambda K is in Z, like that. Okay.

So this real problem – this is the problem that gets you back to that one. It says that the lambda's I chose have to be multiples of M is a thousand. I think multiples of .001, right?

So I comment this out because can't handle it basically. You comment this out and you get – now you get something called the relaxed experiment design because I've relaxed this constraint. You all ready know that because – wait, is that on the current homework? I don't know because we're working on homework's like one and two ahead. Are you doing something with a relaxation and Boolean variables now?

Student: Yeah.

Instructor (Stephen Boyd):Okay. Good. Then you know what relaxation is. Okay. So relaxation is commenting out the constraints that you can't handle. That's really what it is. There was a question? Yeah?

Student: Yeah. I was wondering if you can't handle [inaudible] problem guessing stuff convex? It's like your M is not convex.

Instructor (Stephen Boyd):The only problem – the only thing that's not convex is that.

Student: Yeah. But you can't handle it.

Instructor (Stephen Boyd):You could, but not in right not in what we're doing in the class. No, in general, you can't. So these problems can be very difficult.

Student:[Inaudible]

Instructor (Stephen Boyd):I believe it is, but don't –

Student: Because it looks like [inaudible].

Instructor (Stephen Boyd):Yeah. I think I could make – I'm pretty sure I could make this NPR, but there'd be no – I'd go to Google and I'd type experiment design, NP hard and there'd probably be five papers showing aversions of experiment design for NP hard. I'm guessing, but I don't know. Just to make sure.

Student: Could you also get rid of that constraint if we could use like a steptastic experiment design?

Instructor (Stephen Boyd):Yes. Okay. So that actually is an excellent question and so let me explain how this works. When you do a relaxation – I mention this to you because you're doing one now. Okay. So the way you deal with relaxation is take very good points – good time to talk about it. I'll say more about it later in the class, but when you're doing your relaxation. Now, first let's talk about – let's say what the truth is. The truth is you have a constraint you can't handle and you just comment it out. It's kind of like the truth of why do you least squares? You do least squares because you can and you know how to. That's why you do it, okay. Then you can construct all sorts of stories, which are – some of which are sort of true, not true. And someone says why are you doing this? You go oh, everyone, well, maximum likelihood asymptotically blah blah blah estimator Gaussian blah blah blah.

And they say is the noise really Gaussian? And you go, yeah, yeah, sure it is. Yeah. Anyway, you do – also, by the way, if you repeat those lame excuses often enough you'll actually start to believe them, right? So you'll actually – you start it off as just you couldn't handle it, but what happens is after you've been successful doing least squares for like 15 or 20 years you actually start believing it that things are Ga- anyway. All right. So this is a great example. You're doing experiment designs. You can't handle this, so you simply comment it out and you're gonna solve this problem here. Now, by the way, in this problem you could get very close because you can bound how – how far can you be off if M is a thousand? Right?

So the point is if you solve this problem with lambda being just a real number these are numbers between zero and one. It's a probability distribution. Each such number is at most .0005 away from an integer multiple .001. So you could actually bound that, right? In this case. So you could actually – you don't have to fall back on some totally lame excuse and something like that, but the alternative is actually better because it works universally. It was your suggestion and it goes like this. When someone says what are you doing? Well you say I'm solving this problem here and you go yeah, but this thing has to be an integer multiple of M. And you go oh, no, that's very unsophisticated. I'm actually doing a – what I'm really doing is starcastic experiment. It's like randomized – actually its exactly randomized detector.

And you go this is very sophisticated. This is how – this is like a randomized algorithm, randomized detector. This is much more sophisticated than just committing to 179 of type one, 28 of type two, and so on that add up to a thousand. You go no, no, no, this is much more sophisticated. I'm coming up with a randomized experiment design and I'm gonna come up with a probability distribution on the possible measurements and then what I will do is I will ask for a measure – each time someone asks I can carry out a measurement. I'll flip a coin and do a randomized experiment and I'll use those probabilities.

Now, of course, this is totally lame, this argument, but it often works. So I do recommend trying it and when you relax something and see if you get away with it because it just makes things easier. Did I answer your question?

Student: One other quick one. Can you do better like that?

Instructor (Stephen Boyd):What? With this?

Student: Using starcastic design can you do better than if you deterministics?

Instructor (Stephen Boyd):Can you do better? Oh, yes, because you've removed a constraint. So you always do better in some sense, right? I mean, the problem, the only problem with solving this problem and not the original one, is when someone comes along and says yeah, but I have a thousand measurements. You've gotta come up with some numbers that add up to a thousand and you go, yeah, well, 187.22. And they go what does that mean? I can't do 187.22 experiments of type one. I can do 187 or 188, which is it? So that's the problem. And you go no, but the 22 is more sophisticated because if you had to do a hundred thousand you'd be doing 18722 or something like that. So anyway. I mean, don't – in this case, if M is large you can bound how far off you can be and it's not a big deal.

In the Boolean case you're doing right now, by the way, in those it's often not the case that you can bound how far you're off. Okay. All right. So this is the problem. By the way, these things work really, really well. I should also add the same thing you're doing in your homework now or are doing or will be doing shortly. The same methods work for experiment design. They work unbelievably well. So if you were to – if someone actually said choose a thousand experiments to carry out from this palate of 20 you'd solve this problem, you'd get some lambda's, you'd just round them to numbers. Like, you'd say, okay, 187 of the first one. By the way, that would be a valid choice of experiment design, right?

And it would have a certain value of E, which is a covariance matrix. Okay. The relaxation gives you a lower bound and then you'd say someone could say is that the optimal? And then you could honestly say don't know, but I can't be more than .01 percent off. So it's good enough. So the same thing. You only know what I'm talking about if you've started on the homework, which is maybe not very many people, but anyway, we'll just move on.

Okay. So the other question is how do you scalorize the fact that it's a vector problem? Well, it is a vector problem. It's covariance matrices and, by the way, its experiment design. You get interesting stuff and it's just what you think. I mean, basically what happens is this. I'll just draw a confidence ellipsoid. You make one experiment design and you might get this confidence ellipsoid. Okay. You choose another blend of experiments – well, if you choose another blend of experiments and you get this it's very simple. The second was a better choice than the first. Okay. This is the clear unambiguous section, but, in fact, the way it really works is something like this.

You choose one set of experiments and you get that as your confidence ellipsoid and you can choose another set of experiments and you get this. Okay. And now you can say which suite of – which experiment design is better? The answer is it means that it's total nonsense. It's a multi-criterion problem. Now, by the way, if you're estimating X1, which one is better? Two. Obviously the second one is better, right? If for some reason you wanted to estimate something along this axis obviously – oh, sorry. That axis, then one would be better. Okay. So you have to scalorize this and there's lots of ways to scalorize it and each way to scalorize it, by the way, ends up with a name of experiment design.

The most common one by far multiple – many books written on it is D-Optimal Experiment Design. I'm guessing D comes from determinant, but honestly I don't know. So the D-Optimal Experiment Design you minimize the log depth of the inverse or you maximize the determinant of the covariance matrix. Okay. So that's what you do. As a beautiful interpretation of confidence ellipsoids you are minimizing the volume of the confidence ellipsoid. Okay. So that's the way – that would be the geometric interpretation. There's others. If you put a trace – if you minimize the trace of this which, by the way, has another beautiful interpretation. The trace, by the way, is the expected this. It's the expected value of X hat minus X true norm squared. So if you minimize the trace and that's called A-Optimal design is the other one and there's others.

There's E-Optimal design and they go on and on. Okay. So this is clearly – I mean, it's a convex problem because we're doing relaxed experiment design. It's a convex problem. You know what we should do? We should just do a homework problem on that. Now that they know what relaxations are. Just super simple one. Sorry, we're way behind on that. You should do that where you get the bound and then you do the design. Okay. We'll do that. So here I – actually I'll think I'll quit here, but just we'll look at an example just to see how this works and then I'll talk about this last business.

So here's some possible experiments that you can carry out. So these are the A's. So basically what it says is you're gonna measure – you're actually gonna make a linear measurement and I noticed that these things are farther out. They have a larger magnitude than these. That means that this is a set of sensors here that have higher signal to noise ratio than these measurements. Okay. Now, it should be kind of obvious what you really want to do, if possible, is to have high signal to noise ratio measurements which are orthogonal because if they're orthogonal it turns out it's gonna transfer – when you do the inverse the condition number is small and big will translate to small when you invert the matrix and so on. And so, sure enough, I think we add something like 20 possible experiments – the palate of experiments offered was 20, two were selected.

And the two that were selected make tons of sense. Didn't select any of these because these measurements kind of do the same thing, but with a better signal to noise ratio. So these were opted for No. 1. No. 2 it shows these ones that, sort of, were farthest apart from each other. The measurements that were farther apart from each other. By the way, if you do things like GPS and stuff like that these things will make perfect sense to you, right? These are the – it says you want to take measurements. In fact, you'd even call this like geometric gain or something like that is what you'd call this. So this is what happened.

So obviously in this case you did not need to solve a convex problem to be told that you should take high signal noise ratio measurements instead of low ones and you should, if possible, take ones that are nearly orthogonal. You don't need that. Trust me. You have a problem where you're estimating 50 variables, or ten even, and you have a hundred possible measurements. There is absolutely no way you can intuit what the correct mixture of what the right experiment design is and the result can be very, very good. But anyway, I thought we all ready decided. You'll find out. You'll do one. Okay. We'll quit here.

[End of Audio]

Duration: 77 minutes

ConvexOptimizationI-Lecture12

Instructor (Stephen Boyd):So, let's start with a couple of announcements. I think we're a little bit behind on updating the website of the readings. So it should be kind of obvious. You should read the chapter on geometrical problems. That's kind of obvious, I guess because we're going to start that today. The other is you should read this appendix on numerical linear algebra. It's not very long; it's like 15 pages, just because we're gonna hit this material next week, and what's that?

Student:

[Inaudible].

Instructor (Stephen Boyd):Oh, it's more like 15, yeah, not 50. You wanna read this because the fact is you don't want to hear all this material for the absolute first time when I cover it like next Tuesday or something. So that's just a hint that we're not gonna be moving slowly on this material.

Okay, I want to finish up something about an experiment design which actually relates to our next topic, which is geometrical optimization problems, so experiment design, you recall from last time is this. You're gonna make a number of measurements and you have a pallet of possible measurements you're allowed to take.

That's V1 through VP, parameterize these things, and you're given a budget, for example, just a total number. You're just told, you know make 40 measurements out of this pallet of 50 possible measurements. Which 40 would you take?

And we talked about that last time. When you commit to a set of 40, which is actually nothing, but a set of integers on you know, M1 thorough MP that add up to 40. When you decide on your allocation, you'll be scored on a covariance matrix. That's your error covariance matrix.

Now, that's obviously a vector optimization problem because one person's error covariance matrix could be very small in one direction, big in another and then you could have the opposite, so that's kind of obvious.

The first thing is you have to scalerize. Then the second is there is this issue, it's a smaller issue, it's the issue of a relaxation from an integer problem to a continuous problem, and we talked about that last time.

So the most common scalerization is de-optimal experiment design. So in de-optimal experiment design, you minimize the log determinant of the covariance matrix. So this is a covariance matrix. This is, of course, convex clearly in the [inaudible] here. And this corresponds geometrically to minimizing the volume of the corresponding confidence ellipsoid.

This is sort of like if you're doing a problem involving positive definite matrices where you want it small, log determinant or determinant is something like the least squares there. It's kind of like if you can't come up with and defend another measure, then you might as well just go with log volume or something like that. So it's something like least squares.

Okay. So this is the de-optimal experiment design. Now, one of the things is when we start doing applications, it gets extremely interesting to do things like taking out a problem that comes up as an application. This is an experiment design. This is the optimal experiment design and you actually worked out a dual for this problem.

What's often the case is that a dual of a practical problem will actually turn out to have an interpretation, which is another practical problem, and it can be one from a totally different field or something like that. After the fact, everything will make sense, but it will not be remotely obvious at first that the two things are connected.

So in this case, a dual, which I'll work out shortly is this, so these two are duals, and of course, what I mean by that in the loose sense, what I mean is that I transform this slightly, formed a Lagrange dual, and then made a few more small transformations of that one.

Maybe I solved a [inaudible] variable. Maybe we'll get to that. This is called – so these are duals in the loose sense. However, they are duels in the following sense. Any feasible point here provides a lower bound for this problem period.

Okay, there's an easy transformation – if you solve this problem, you can solve that one and vice versa. There's zero duality gap here, absolutely zero in this case, because the weak form of Slater holds. Slater condition says you have to have a point that's strictly feasible.

Well, sorry, the strong form of Slater's condition hold here because I'll just take all the [inaudible] to be one over P, okay? And then the strong form holds, but anyway, so zero duality gap between these.

Now, this problem actually has a very simple interpretation, [inaudible] as a constant, so you're maximizing log det W or if you like, you're minimizing say log det, say, W inverse. That's the log of the volume with a constant multiplier of the half or something like that.

And then also an additive constant, but then an additive constant and a factor of a half or something like that, that's the log of a volume of an ellipsoid, and this would be K transposed W inverse parentheses inverse DK less than one, and this is the problem of computing the minimum volume of ellipsoid that covers the points VK.

Okay, so I mean this sort of, by the way, this problem comes up a lot as, well. In fact, I'll mention statistical application of this right off the bat. Actually we'll get to some of that in the next lecture, so maybe I'll wait. So what this says, and you can work out what complimentary slackness is. Complimentary slackness says that if [inaudible] is positive, that means you actually carry out an experiment of type is [inaudible] three is positive, that means you'll actually use B3.

You'll actually do a positive number of those experiments. Then you must have one minus VK transposed W VK 0. That means VK is on the boundary of the smallest enclosing, the minimum volume enclosing ellipsoid. So the picture is this – a very simple one. We looked at this last time, but very briefly. You have something like 20 choices of experiments to carry out here.

Okay, now these experiments are pretty much the same as these, I mean there's a negative sign; it doesn't matter. But the vector V is smaller and that corresponds to, you know, twice these measurements are the same as these and have twice the signal to noise ratio.

So it would be foolish to choose any of these and indeed, none are chosen. Now, over here, all the weight is put on these two, and this sort of makes sense because the only difference among these is the angle and given the choice of measurements, you would take measurements that are to the extent possible orthogonal. Okay.

So it's actually short of chosen, this point and this point and this is actually put a weight of .5 on each of – meaning if you were do to some, if you were to give an budget of experiments, half should be here and half should be here.

That's what it means and it kind of makes sense. There would be no reason to skew them because they're approximately, they're 30 degrees apart or 35 degrees. I mean they're as close as you're going to get to orthogonal. They're the most mutually – they have the largest angle and so they give the most mutual information or whatever, not any information [inaudible], but they're maximally mutually informative if you take ones that have a high angle, so that's what these are.

Now the interesting thing is the dual has a perfect interpretation. You take these 20 points and you compute the minimum volume ellipsoid that covers them centered at zero and that is, in fact, the one drawn here, and sure enough it touches two points. These two here, and these two actually are gonna be the ones you use in the experiment design, so that's another way to understand in the experiment design you're choosing points that are far.

Now, it's not as simple or as – it's not as simple as choosing the points that are farthest. Now, that would be a greedy scheme to choose the experiments that have the highest signals to noise ratio. That's not the case because in that case, if one sensor had a very high signal to noise ratio, you'd basically only take that one. You actually compute the minimum volume ellipsoid that covers these points and then these give you the other ones that fill in other directions and so on.

Okay. So that's that. And I think I'll skip the derivation of the dual, other than just to mention a few things about it. It should be straightforward for you now. There's no way you could look at this and just say, "Oh, okay." But, you know, it's something that in a few minutes you should be able to trace through. It's quite straightforward.

The only interesting thing here, I don't know if we've done enough problems on this or we've gone over this enough, is actually when you have a matrix problem like this, you actually have to calculate the gradient of the log determinant of inverse of a matrix.

That's complicated. The gradient is a linear function on matrices, and you actually have to, I mean don't just take, the formula is basically it's minus X inverse. But you have to be very, very careful to understand what that means.

That's actually covered in an appendix in the book, so I'd recommend reading that. In fact, let's add that to our list, and I don't remember what appendix this is. Okay, so it's the one that covers, let's say, that, what does it mean and so on and so forth.

Okay. All right, so how do you form, when you reformulate it this way and have a matrix constraint here, it's quite straightforward to introduce a Lagrange multiplier. Normally, it would be new transposed times you know, X minus B. Here it's the matrix equality. The transpose is an inner product. In inner product for matrices is traced in general [inaudible] Y. These are some metrics of trace XY. So you write this as Trace Z times the matrix residual here.

Okay, so that's the origin of that. Otherwise everything else kind of works out the same and I'll skip the details. Just to say the bigger picture is, basically for any problem you look at, and have time to actually think about carefully, it's actually worth your while to work out the dual because the dual often has a very interesting interpretation, give you some better idea about how it works and all that sort of stuff.

Okay. We'll move on to our last sort of generic family of applications and these are in geometry, so geometric problems. These came up actually all the time, and we'll start with some of the ones that are least obvious, the extreme [inaudible] volume of ellipsoid. So we'll look at that. These are some of the most interesting and least obvious applications. In fact, it's the least obvious what you can do in these cases.

So the L' owner John ellipsoid of a set C, which need not be convex, I mean, at all, is the minimum volume ellipsoid here that covers the set. Okay, so that's the definition of the L'owner John ellipsoid. Actually it's unique. We'll actually by formulating it as a convex problem, we'll prove that right away.

So here it is. We'll parameterize the ellipsoid as the – this is the inverse image of the unit ball under an affine mapping, okay? And that's a parameterization. I should point out there's at least like four or five totally different parameterizations of an ellipsoid, right and you can write it as a quadratic form. The forward image of an affine mapping of the unit ball, you can write it as the sub level set of a quadratic function. You can write it all sorts of different ways.

Which of these methods you use to describe an ellipsoid, to parameterize the ellipsoid will completely change the convexity properties, and so when you have different problems, you're gonna have to choose one of these. So it's not like this we could not, this will not work out, if we parameterize for example, it this way, X minus XC, a set of X, such as that.

X minus XC transposed, say P inverse, X minus XC is less than one. That's another parameterization of an ellipsoid. And the data structure to describe this ellipsoid is a center point, XC, and a positive definite matrix P. Now, it's easy to go back and forth between this and this. It's just linear algebra, okay. It's easy; however, when you formulate the problem here, you'll end up with a non-convex problem for the L'owner John problem.

Here it's gonna end up being convex, so you have to choose the right parameterization. It makes a great deal of difference which one you choose. So this is the inverse image of the unit ball under an affine mapping. It's an ellipsoid. And you know, it's not hard to go from this to this. It's not a big deal. You know, A transpose A is P or something.

Now, in this problem here, it turns out you can assume without loss of generality that A is positive definite. And that's not at all hard to show. I mean one is this; it's got to be non-singular. If A were singular here, then it would, of course, have a null space and in fact, this would be called a generalized ellipsoid or a cylinder because anything in the null space it would have an infinite dimension here.

That's what this, it would have a directional along which was infinite. It wouldn't be bounded in that case. Nothing wrong with that, by the way. Sometimes that comes up. That corresponds to A being positive semi-definite and not positive definite.

Now the question is why can we assume A to be positive definite? Does someone have any suggestions about why? If someone gave you an A here that was non, not symmetric, not positive definite, how can I compute, how can I determine the same ellipsoid with an A that is positive symmetric and positive and definite? How do you do it?

Student: You just flip the side of the eig indice.

Instructor (Stephen Boyd):You could flip the side of the eig indice. A might not even be symmetric, but you're on to the right track. What would you do, so let's right the SVD of A. Let's write A here as U, sigma, V transposed times V less B, this is less than 1, all right.

So this is the set of V for which this is true; right? And I need a suggestion. This is A, and I want to write this in a new form where sort of the new A and the new B are symmetric, A is symmetric positive definite.

So they're all writing the skeleton. So I need a suggestion.

Student:[Inaudible].

Instructor (Stephen Boyd):What's that?

Student:[Inaudible].

Instructor (Stephen Boyd):You're gonna choose what?

Student:U equals V.

Instructor (Stephen Boyd):Okay, choose U equals V, yeah, I mean that's kind of what we want to do, but you can't say that. I mean somebody just gave us this description with A and B. Their A is not symmetric. So we don't have the right to say choose U equals V. They said, "No, no, no, this is the A I'm giving you."

So one thing you can do is this. This is a vector and if I multiply that on the left by any orthogonal matrix, it doesn't change the norm. That's the same. So I could multiply, let's just do this in our heads; let's multiply this vector on the left by first U transpose.

Student: Mm-hmm.

Instructor (Stephen Boyd):And then by V, and because I was multiplying by matrices, the first was whatever, so we're gonna write it this way. V, U transposed times U sigma V transposed V plus V. Everybody cool with that. I'm mean that's identical. This is equal to that because that's an orthogonal matrix. Everybody cool with that? Okay, and what do you have now?

Now we're done.

Student: You have V.

Instructor (Stephen Boyd):Yep, now you have this and everything's cool. This V, sigma V transposed plus then something here which is V, U transposed B –

Student:[Inaudible].

Instructor (Stephen Boyd):Yeah, I did, yeah, somewhere; where here?

Student: Yes.

Instructor (Stephen Boyd):There we go. Thank you. Okay. And I'm gonna call this A tilde and I'm gonna call that B tilde, and I'm done because I just took the original ellipsoid. I mean this would be the code you'd write at the top of something where you decided in a group of people doing whatever development that the data structure of an ellipsoid is the inverse image of an affine mapping, okay?

However, you don't put in the specifications that the matrix should be positive definite. Okay. Somebody passes in this, but for your method, you need this. This is the stuff you write at the top that translates it to an equivalent one with positive definite A. Okay. Everybody okay on this?

Student: You could have [inaudible] right?

Instructor (Stephen Boyd):No, it's a singular value decomposition. It's not eig in value decomposition, singular value decomposition and it's non-singular with the original matrix A, so they're all positive, so we're cool.

Student:[Inaudible] A is not singular, then A tilde is. If A is singular –

Instructor (Stephen Boyd):Oh, pardon me, right if A is square and non-singular and therefore U and V are square orthogonal – they're orthogonal matrices.

Student: The whole point of this was that A was not squared.

Instructor (Stephen Boyd):No, no, no, no. The whole point – A is squared. Good question. Actually you can do this, by the way, if A is not squared, too, but you'd have to – you can do this in that case, too.

What A has to be in that case is, well, let's see, it has to be fat and full rank or skinny, mini fat and full rank, I think is what it – it has to be; sorry, skinny and full rank is the most general. If you want to allow people to describe an ellipsoid in this way in the most general way, A, I believe can be skinny and full rank. That'll do the trick.

Student: You're not losing information [inaudible].

Instructor (Stephen Boyd):Yeah, exactly, so in that case we're cool because and in everything I did here was kosher, so let's just, you know, in that case you have to check, go back and check that U has the right – well, I guess U transposed to U was always cool. So, okay. Question?

Student: Why did we embark on this journey in the first place?

Instructor (Stephen Boyd):Well, I haven't said yet.

Student: Not the journey of the ellipsoid, the journey of positive symmatrizing the matrix?

Instructor (Stephen Boyd):Now, would we embark on a journey – well, we shouldn't answer that. I was gonna say, "Would we embark on a journey that was not needed?" The answer is yes, we do that every day, but they're kind of weird things. I don't usually have it in writing, those weird little journeys and side trips. So obviously, this is gonna come up, right? Indeed it will in two lines, so, but thank you for putting the posting the sign there which is, you know, why are we doing this? Okay, so let's move on.

Student: Why does it have to be – so you're assuming A is not singular?

Instructor (Stephen Boyd):That's right – oh, why? If A is singular, this doesn't describe an ellipsoid; it describes something called the generalized ellipsoid. It's called a cylinder, in fact, because it will be infinite in some directions, in fact, specifically in the null space of A. Along the null space, any point in the null space of A can get infinitely big and satisfy this, and everything's cool; right.

So in that case, so that's not an ellipsoid. By the way, such a thing has infinite volume. And if we're minimizing volume, it's not gonna be something. It's sort of infinitely bad; its not gonna be of interest. So that's why.

Okay. Now, I'm gonna use this parameterization and yes, I wanted a symmetric positive definite for a reason and now you know why. Here is it. By the way, you see this problem here; this problem is actually correctly the L'owner John ellipsoid problem even when A is not symmetric; okay. It is because I assure you, log det A is gonna be the volume, it's gonna be the volume while multiplied times the volume of the unit ball on that space. If that's gonna be the log of the volume, actually it's already added to that of this ellipsoid period, or one over it is, all right.

So, however, the log of the determinant of some non-symmetric matrix is some insane function that is not in general convex; in fact, it's never convex – just not. So however, if I assume A has symmetric positive definite, then this problem here is convex and that means we can solve it.

So that's it. So here I minimize log det A inverse, and you have to check about the volume and things like that, but I mean that does transform – the volume transforms by the factor det A inverse here.

Now, to say that – I mean I can rewrite this way. I don't have to write it this way. I can write this, right, for all B and C. This just basically says that for every point in C, you're in the ellipsoid. You're in the ellipsoid if and only if AB plus B, when you take the norm, is less than 1.

To say that the ellipsoid covers C is to say that for any point in C, norm AB plus B is less than or equal to 1. So that's this statement. Now, by the way, we know immediate that it's a convex problem.

Some people, by the way would glorify it and make it very complicated and call this a semi-infinite problem. Why? Because that is a convex constraint – in fact it's a two-norm constraint on the variables. By the way, the variable here are A and B. So actually reading this kind of stuff requires serious concentration because it’s a very standard convention that symbols like A and B and C and alpha and beta are parameters.

And variables are things at the end of the alphabet like U, V, W, X, Y. So this is totally turned around and the variables are A and B and the parameters – well B is a dummy parameter, but nevertheless.

Okay, so you have to read this correctly. That is a two-norm constraint. It’s nothing more, but it’s 1 for every point in C. Oh, if C is infinite, this is a so-called semi-infinite problem. Okay. Now, you may not be able to have a tractable representation of this semi-infinite constraint. In some cases you can. I mean one is a finite set, so here I give you a bunch of points and you have to calculate the minimum volume ellipsoid that covers them.

Now, in this case, it’s easy to say this because you just write that. This is a convex problem. So now you know something. And of course, it’s not just the L'owner John ellipsoid for the set consisting of these points. But in fact, the convex [inaudible] those points and that’s a polyhedron described by vertices.

So you just learned something which is by no means obvious, and that is that if someone gives you a polyhedron defined by its vertices, then computing the minimum volume ellipsoid that covers those points is a completely tractable problem. Okay. Everybody got that?

And in case you think, you know, sort of this is obvious and everything works and all that, watch out, because if I give you a bunch of points like this and I give you a polyhedron described by the vertices and I ask you for example, to solve a variation on this which is please compute the maximum volume ellipsoid that fits inside it, that’s actually MP hard, okay?

So I’ll show you, of course, I’ll focus on the ones that actually we can do, but watch out. One step off this path and things get very complicated and these are not obvious facts. They’re just not obvious period. Okay. Let me mention a couple of things about this one. This has huge applications, sort of choosing maximum volume ellipsoid around some points.

And let me sort of just give you one or two, they’re beautiful applications. Here’s a really very nice one. Suppose you just simply have V1 up to V, doesn’t really matter, you know, 10,000. It doesn’t matter. These are measurements – they’re vector measurements. They’re in our tenth.

That’s it; they’re vector measurements. And what you want to do now is check are there any measurements that like stand way out. In other words, are there outliers. That’s the question here. Are there gross – are there some measurements here that are big and somehow don’t fit with the set; okay.

So how would you solve a problem like that? It's vague, comes up all the time. I want to identify outliers in a bunch of samples. What would I do? Make some suggestions, actually. Let's start with unsophisticated ones. So what would you – what would be the first thing you might do?

Student:[Inaudible].

Instructor (Stephen Boyd):Sure, yeah, absolutely, so the simplest thing is you might – let's subtract the mean of these from here. So no they're sort of centered roughly around zero. That would be reasonable. Then actually at that point, you might just look at the norms of all these things, and if all the norms were on the order of you know, half, one, two and a handful of them have the norm of 50, then there's your outlier problem solved, right?

So basically you have a cloud of points like this and a couple of just way out there, okay. Everybody agree? So that would be the first thing you might do. Then you might say well, no, no, that's not right. What I'll do is I'll take these points; I'll subtract the mean. That will center them and now I'll calculate the empirical covariance matrix of them. That gives you the ellipsoid that kind of covers the cloud the best. Everybody cool on that?

Then I might even change coordinates so that ellipsoid was uniform, was a ball. That's a very hard change of coordinates – the one I mentioned before. I don't recommend it without a lot of practice.

So you change coordinates and the ellipsoid becomes a ball, came out right. And now it means roughly speaking the data kinds of varies equally and now if you start seeing points that are way out these could be your outliers. Everybody okay on that?

Now, the problem with that is that that one – well, that would actually, that would, sorry, that would actually work – that would probably work okay. Turns out a much more sophisticated method than that is actually to compute the minimum volume ellipsoid that covers these, okay.

The points that are on the surface of that ellipsoid actually are ones which you would declare as candidates for outliers, okay? And in fact, the way this often works is you take those points and you remove them and then you redo whatever you were doing, some lead squares problem, some singular [inaudible]. It really doesn't matter; you do the processing.

What you're looking for is something like this. You want to remove a small number of points, do whatever signal processing or statistics you were doing before and have the answer all of the sudden get way better; right?

So you want to remove eight points from the set of 10,000, do some signal processing and all the sudden have like an excellent fit. This is a hint that at least that those eight included the ones that were messing you up before. Everybody see what I'm saying? That would be outlier [inaudible].

Okay, so this is called – the method. This is called in statistics – it's called ellipsoidal peeling. It's the greatest phrase, right, so you have a giant pile of data. You stick an ellipsoid around it and you remove those points. Once you remove those points, by the way, if you're quality of estimation of whatever signal processing or statistics you're doing didn't get like really good, you do it again.

You've removed those points and the ellipsoid shrinks down to fit the next one and this is called ellipsoidal peeling and you do this in waves or whatever until you like what you get or until you have no data points anymore, in which case you went too far. Yeah?

Student:I was gonna ask a question. Let's say you do it, you peel like two or three layers and you don't see any substantial change. Does that tell you that maybe, you know, you didn't have any variable outliers before?

Instructor (Stephen Boyd):Yes, it might tell you that, yeah. Generally, when that happens, when you do a couple of cycles of ellipsoidal peeling and it doesn't get better, generally you don't admit to people that you ever did that.

So yeah, you just go back and say, "It's not working." That's what you do, although you're prepared if someone says, "How do you know that there's not – maybe there's just like five or six outliers, just like bad data points that are completely messing up your, you know PCA or whatever you're doing, or your imagery construction or whatever it is?"

That's how, and then you'd say, "Well, you know, I did do some, I did some ellipsoid peeling. And nothing got better." By the way, we've already talked about it last time, but what's another method for dealing with outliers – I mean just grossly? What?

Student:[Inaudible] different norm?

Instructor (Stephen Boyd):Yeah, you'd go to an L1-type norm or a Huber norm, or something like that – Huber is not a norm, but you'd go to a robust norm. By the way, you could do the same thing there, the peeling thing.

Student:[Inaudible].

Instructor (Stephen Boyd):Take all of them out. You take all the ones that are on the boundary. If you don't want to take as many out – if this is 10,00 and these are an R10, I forget how many are gonna be on the boundary, but it's generically something like 50, so 50 out of 10,000.

Student:[Inaudible] do it again.

Instructor (Stephen Boyd):Yeah. I do minimum volume ellipsoid. I think generically about 50-55 or something like that would be on the boundary. By the way, if you take out the 55 and all of a sudden, like your image appears and that's like somebody's head. You're doing medical imaging, then you know the following, you know that A, there were some outliers, No. 1 and No. 2, they were within the 55 year removed.

Now, if you care, you can go back and start putting points in one at a time until the head goes away, okay, and then you say, all right, that, you know, so if you care, but on the other hand if you have 10,000 measurements and you throw away 55, you know, it would seem unusual if suppose instead you threw away 22 outliers and whatever, 33 non-valid data points.

Loss of 33 points, I mean I'm just talking – this is very hand-waving, but loss of 33 points out of 10,000 or I should say 9,950 is not gonna really make a big difference, so that's what you do, so you can add them back in or whatever you like.

These are actually fun methods. They can be shockingly effective, and I should add actually on the whole issue of outlier detection – I should make one comment about it. When an outlier is like totally off scale, anybody can do it. So if like you're getting a bunch of measurements, you know, and a bunch of them are like you know, 1, 2, 3, minus 5, minus 6 and all the sudden, one is like 1E plus 37, okay. You don't need advanced methods to detect that as an outlier.

That's because a bit flipped in a floating-point representation or something, you know, in the exponent. You don't need this for that, okay. On the other hand, if the outliers, you know, basically don't – if an outlier is just like it's just off a little tiny bit. It's just like some like 1 over [inaudible], and you know, it's not Gaussian, but it's not huge, not big enough to mess up your algorithm or you have enough data that it gets averaged out nicely, then you don't have to remove it.

But there's a band of situations right in the middle where being smart actually is gonna pay off. It's that band in the middle, it's when the outliers – the outliers are small enough well they don't really hurt anybody. If they're big enough, anybody can spot outliers, but there's a pretty big band right in the middle where being smart is gonna make the difference between successful signal processing statistics or whatever this is, whatever application it is and not. So that's just my comment on this.

I guess this is silly – that's the case for all problems, this band. Yes.

Student:[Inaudible] continue all the measurements. It's just to [inaudible]?

Instructor (Stephen Boyd):Yeah, so you're removing outliers and so the argument, so first I'll tell you the truth. The truth is say you're allegoristic, so it cannot be defended; however, you can say in its defense, the following: if I simply moved, it removed the values of V with large norm, that would be something that is not scale invariant.

If you were to rescale the data or transform the data, I would remove a different set because I can make a difference out of points look big. Okay, the interesting thing is if you do minimum volume ellipsoid, that's actually affine invariant.

You'll select the same point because think about it, if you have a bunch of points, and you transform them, you know, like by any linear mapping, you wriggle them all around and you re-calculate the minimum volume ellipsoid, you get a commutative diagram. So the minimum volume ellipsoid that covers a set of points is affinely invariant. It commutes with an affine change of coordinates.

Therefore, if someone says why are you going to the trouble, why not just take the norm and remove the ones with the largest norm, right, you would say, I'm doing something a little more specific, I'm minimizing the one that is large, but my measure of large doesn't depend on the coordinates used or the scaling used; it's more sophisticated.

Funny, if you draw that story out long enough, people won't notice that it actually doesn't answer the question, which is why. It's merely a justification. Okay.

All right, so that's an application of this. There's sort of a dual problem, that's maximum problem inscribed the ellipsoid. So here I have an ellipsoid – we'll get to this. Here we're gonna describe the ellipsoid, by the way, as the forward image, not the backward image of – it's the forward image of a unit ball under an affine mapping.

And again, I can assume, I won't go through the details that B is positive definite here. Now the volume is proportional to det B and the problem is to maximize log det B subject to this.

What this says is another way to say this. It basically says that the [inaudible] over the norm ball of BU plus D is in sync, so that's another way to say it. If you want the semi-infinite representation, you would write BU plus D is in C for all norm U less than 1.

Now we know, by the way, that is a convex constraint on B and D. Why – because for each U in the unit ball, this is a convex constraint. And remember what the variables are here. It's B and D. This is not obvious. U is fixed. That's an affine expression. It's affine in B and D. Actually, it's linear in B and D, in fact.

And so you have the constraint that a linear function should be in a set. Sorry – this is the case of C is convex; sorry. If C is convex, this is a convex constraint. Okay.

Now, you still have to be able to evaluate this to make this interesting, and there's actually a very simple case when you can and that is when this simplest case is when this thing – when the set C is a polyhedron because I can tell you whether an ellipsoid described this way lies inside a half space.

That's easy. And the way you see that, that's an easy calculation. We want to know is it true that BU plus D – let's write a half space, so the half space is the set of the X such that, let’s say, F transposed X is less than G, okay. That’s our half space, and I want to know is it true that BU plus D is in H or all U norm less than 1. That’s the question, okay.

And the answer is easy. Yeah. We can calculate that. The question is whether F transposed BU plus D less than or equal to with a question mark like that for all U less than 1. Everyone agree?

Now, by the way, we’re focusing on this as a function of B and D. We’re getting there. Now, the second term is this, F transposed D. It has nothing to do with U like that, and here I'm gonna write this in a different way. I'm gonna write this as B transpose F, transpose U. And t his should be true for all norm U less than 1.

Now, if someone walks up to you and says, and you form an inner product with a vector and says, “You know, I need the inner product of a vector with U, U ranges over the unit ball. We know exactly what these numbers range over. They range between plus/minus the norm of B transpose F.

Okay, in fact to be quite precise, that’s [inaudible], but in the general case, general norm, it’s plus/minus the dual norm, the dual of that norm. So the maximum value this can have is this period, and that is with the choice U equal B transpose F divided by norm B transpose F. That’s the worst thing that can happen. So these are all if and only ifs. I haven’t drawn them, but it’s that. Okay.

And then you have that, and that’s it. We’re done. Okay, because now this is tractable. That is a convex function of B. This is a convex function of something happened there – D, thank you. That’ a linear function of D. The whole thing – this is clearly convex and B comma D. Okay, so that’s the condition and that’s what we have here.

Maybe I have it wrong here, or did I assume B is symmetric – yeah, sorry. I assumed B is symmetric, so the transpose isn’t needed here. Okay, so I get that. By the way, just let me point out, you’re used to it. I don’t know – we’re six weeks into the class. There’s something like that. It’s not a big deal to look at that and say that’s a convex problem in B and D.

These are things that if you don’t know this material, that looks like a hopelessly complicated problem, I mean hopelessly complicated just because, you know, it’s got the log of the determinant and all that. You wouldn’t know anything. This is non-differentiable; you know that. The norm – that’s not norm squared; that’s norm, right, but you look at that and say that’s like a second order of constraint. It’s nothing, the second order of constraint.

So all right, this allows you to calculate, to compute efficiently the following. This says that if you have a polyhedron described this time not by the vertices, but by the inequalities, so basically you have these things, AI, like that. You can now calculate efficiently the maximum volume of ellipsoid that sits inside it; okay.

And that you can do efficiently. Now, interestingly, if I take a polyhedron described by inequalities and I ask you to calculate the minimum volume ellipsoid that goes, you know, the minimum volume that covers it, that's MP hard, again. So you have to watch out with these things. They're not – I'm just casually mentioning them here. This is not simple. Things that are – you know, easy problems and just minor variations on them are very hard.

Question?

Student: Is the MP hardness because you're using the representation of the polygon, which doesn't actively constraint?

Instructor (Stephen Boyd):Something like that.

Student:[Inaudible].

Instructor (Stephen Boyd):Yeah, MP hardness is actually very easy to see. It turns out, you wanna hear something hard, is just the following. Forget finding the minimum volume of ellipsoid that covers a polyhedron described by inequalities. They this much easier problem. Suppose someone just asked you the following.

Here is a specific ellipsoid. Can you please tell me if it covers my polyhedron? That's the feasibility – not even the feasibility problem. That's not finding the polyhedron that's feasible. It's basically check if a specific polyhedron is feasible.

That's MP hard. Turns out it's maximizing a convex quadratic function over an ellipsoid. And I can easily embed that to all sorts of MP hard problems.

Yeah, but I don't know that I really answered your question. I don't' know that I really have a good answer for why is it easy to go from – if the data structure for the ellipsoid is vertices, why is it easy to calculate the minimum volume in covering ellipsoid, but hard to calculate the maximum volume 1 inside. And why does it reverse here? I don't know; I don't really have a good story for that, so I don't know. Question?

Student: So given this description, for example, we have this [inaudible] can you actually envision and find vertices?

Instructor (Stephen Boyd):Excellent question. Now, of course, if you could efficiently go, if you could transform between these two data structure to describe a polyhedron, then of course, right here, right now, we would have shown P equals NP. Now that – I'm not ruling it out. Could happen someday, maybe around Week 8 or something like – no. But it – no, I'm glad you pointed that out because in fact, there's a catch there.

So the question is given a polyhedron described by inequalities, can you describe it by its vertices, and the other way around? And in fact, these are just two different data structures, to describe the polyhedron, right. That's no problem. Now it's interesting because, for example, ellipsoids, we were just talking about multiple data structures describing ellipsoid. Those are all equivalent. They’re all about the same size. They all involved like some sort of a matrix and they involve a vector.

You know, roughly, you know, in that – you know, transforming from any one to any other is just linear algebra. That’s all very straightforward and everything is polynomial. It’s all like N-cubed or whatever, right.

So those are sort of equivalent, they’re different, but equivalent parameters; they’re polynominally equivalent parameterization. Now, in general, there’s some pretty shocking things, I’ll just mention some of them. If you have a polyhedron described by inequalities, it is possible to list the vertices. It’s possible. There’s algorithms that just work, just lists them.

Here’s the problem. The number of vertices is exponential in the problem size. Okay, so even, you know, if you have a – I mean thing s you can just write down in a minute, you could just write down, for example, take 100 dimensions, 200 linear inequalities, okay.

So that’s nothing, right. You can solve an LP of that size, well, in three weeks, you’ll write a code for it, and that can be solved literally in milliseconds. Okay. So you know, everyone has the right visualization skills. When I say minimize [inaudible] where A is 200 by 100, we’re all visualizing the same thing. We’re visualizing a polyhedron and a direction and you kind of find the vertex or whatever that’s most in that direction. Everybody got that?

Now, here’s the problem, the number of vertices of a polyhedron in R100 defined by 200 linear inequalities is absolutely immense. I mean it’s some number that, you know, it’s on the order of like 100 factorial or something like that. I don’t know what it is, and by the way, if I'm not quite right, and that’s not equal approximately to the number of sub atomic particles in the universe, I have to change those numbers very slightly and it will be. Trust me, okay.

So that’s the problem. It’s the going from inequalities to vertices can lead to a huge increase in the size of the problem. That’s the first problem. Now, by the way, for certain cases it’s not. I mean if everything is an R2, anybody can figure out a way to transform between these two representations, vertex versus face or in that case, edge.

Anybody can do it in R3, as well, and if you go to Google and type things like transform vertex face blah, blah, blah, you’ll find tons of papers, but it’ll be for specific cases like R2 and R3 is what you’re gonna find.

You’ll even find for example, in some RKs that they’ll be polynomial algorithm. But K has to be fixed, the dimension, so you’ll even see that. By the way, the other way has the same problem going from one to the other. So anyway, let me summarize all of this because there was a weird, random walk, so let me summarize it.

The change in data structure to describe the different parameterizations of ellipsoids, that's casual, it's linear algebra. It is just linear algebra. It has taken, you know, some square roots, a couple of singular value, inverses, nothing, okay.

Changing the data structure described a polyhedron from vertex to inequality description, that's not casual. So that's a very important thing to note, I mean unless you're in R2. In R2, there doesn't seem to be any real problem anyway because your eyeball can solve all of them and it hardly matters; okay.

So I didn't even consider those to be real problems.

Student:[Inaudible] the problem we just solved earlier before we got this example, not correlate with what you're just saying about covering [inaudible] finding the minimum [inaudible].

Instructor (Stephen Boyd):In the problem before, you're given the vertices and you're asked to find the minimum one. That's easy. In this problem, you're given the faces and you're asked to calculate like the minimum, maximum volume of ellipsoid that sits inside it.

Those two are the easy problems. You flip those two, or re-match them and they're actually essentially infinitely hard, not only infinitely hard, but they're basically you don't even get to this point. Just if someone just alleges an ellipsoid covers a polyhedron, you can't even verify it.

You wouldn't know, let alone optimize over such an ellipsoids. So there's a – I mean these are subtle and you have to look at them carefully, think about them. Okay, now there's an amazing fact here and it's actually beautiful – it's the sort of fact about geometry.

Its' absolutely beautiful and it's this. If take a convex bounded on [inaudible] interior set and compute the L'owner John ellipsoid so that might be say this polyhedron here and you compute the L'owner John ellipsoid, so you find the minimum volume ellipsoid that covers it, okay. And that's this one.

The following is a fact, comes right out of duality. I won't derive it, there are derivations in the book, comes right out of duality. If you take that ellipsoid and you shrink it by a factor of N, then you're guaranteed to be inside the set.

Okay, by the way the same is true here. If you take the maximum volume of ellipsoid that fits in a set and you grow it by N, it covers the set. Okay, now, you might say N is a big factor. Well, yeah, but at least it's a factor.

And it basically says the following, by the way, if you split the difference, I can say it this way, I can approximate any convex set, you know, bounded [inaudible] interior and that kind of stuff, any convex set in RN within square root N.

That's what it means because I would split the difference here, right, I'd shrink this ellipsoid by square root N, and then I'd get one where if you scale it by square root N, you cover, you shrink it by square root N, you're inside.

By the way, if this set is also symmetric, you can sharpen it to square root N, in which case I can even say I can get it into the one quarter approximation. The importance of this is extreme, both theoretically and practically and let me say why. This basically says that ellipsoids are universal geometric approximators of convex sets.

That's what it says because I can bound sort of – you know, I can get a bound and there's a fixed factor whereby I can shrink and expand and fit inside and outside the set. Everybody got that?

Now, that's gonna come up; it's gonna have lots of implications, but what about this. What is someone says, how about a ball. A ball doesn't have that property obviously because for example, here's a convex set. I can make it as skinny as I like, but sort of the minimum size ball that fits this is quite big and the minimum sized ball is like that and there's no bound on these.

So I could hardly say that I cannot approximate a convex set in RN by a ball. How about a bounding box? How about a rectangle? Can I do it by a rectangle? The answer there is N, sorry, that's got N parameters. The ball has only, well it's got N. A bounding box has like two or three N or something.

That's not enough either because in this case, for example, the bounding box can look like that. That's the bounding box, the smallest box that covers the set. The biggest box that fits inside is gonna be tiny period. And there's gonna be no bound on the ratio.

So ellipsoid is sort of the first time when you get enough sort of degrees of freedom to carry out universal approximation of a convex set and let me just ask you another one just for fun. How about simplexes? That's the convex held of N plus one point. By the way, the answer is not obvious here, so you're just gonna have to guess, but I'd say you can do it, so how about simplexes, what do you think?

Can you approximate any convex set by a simplex; what do you think? You can, yeah, you can. There's enough – you have enough free parameters in a simplex to do that and it's easy because you approximate it by an ellipsoid, transform the ellipsoid to the unit ball or something like that and then stick a simplex around that and you can even bound the ratio or all that. It won't be N; it'll be N-squared or something. It doesn't matter; it'll be something. It'll be bounded. Question?

Student:[Inaudible] rectangle instead of –

Instructor (Stephen Boyd):Oh, you mean a rectangle that can rotate? Yeah, then you can do it. Right. Then you can do it. Of course, we don't know how to calculate such a thing, but if you could do it, that gives you enough degrees of freedom to cover it. Okay.

So ellipsoids are not just sort of convenient things that describe shapes of things. It’s actually very worthwhile to understand and remember that there’s a sense in which they capture sort of the, let’s say, yeah, something like the first order approximation of any convex set, as an ellipsoid.

By the way, let me mention one more property of this. Suppose you took, let’s say, ee263. Let’s just suppose you did. Okay, and someone says, “What are you doing?” Or you take a control class or for that matter you take a statistics class. And someone says, “What did you do all quarter?” Everything you did involved quadratic norms, right? You’d have quadratic functions.

In control you’d have X trans plus QX. In statistics, it would be hidden by something else or whatever, and someone would then ask a question like, they’d say, “What did you learn in two seasons?” “Well, I can minimize a norm.” And they’d say, “A two norm, really and then they’d say, “Does the sum of the squares actually come up in your prop? Is that really what’s important in your helicopter design?”

And you could look at them and see how gullible they are and if they look gullible you could try to go, “Oh, yeah, absolutely.” Yep, sum of the squares of the vertical acceleration, pitch rate, absolutely.” That can’t be more than 1.3. If they go for it, the conversation is over. If they don’t go for it, then you’d say, “All right, I’ll tell you the truth. No. We don’t care about sums of squares of things. It’s just because we can do it and because that’s the only class I took so far.”

So that’s generally the truth about [inaudible] squares. Okay, so all right. How does that have to do with this? Well, let’s suppose that there is something you really care about like maneuverability or ride comfort of something. It doesn’t really matter. And suppose what you do now is you go around and you run huge tests, wind tunnel tests. You have pilots write things out and say whether they like the ride or not.

And anyway you end up with some weird set that looks like, you know, some kind of weird convex set that looks like that. That’s like nothing’s happening and this is sort of a set of ones judged by, you know, wind tunnel simulations, actually getting pilots to ride it and all that and they’ll say this is okay. If you’re – this is of course, in a multi-dimensional space, but you know, and if you’re in that set, you know, one of these things is like pitch rate, one’s this, one’s the RMS. It doesn’t matter.

If you’re in that set, everything’s cool; that’s it, okay. That’s the real set, and this is not an ellipsoid. Let me tell you it is not an ellipsoid period. That’s the real – this is the set, okay.

Now, this actually – now what you know now allows you to do something that would let you sort of go back and in a posterior way justify 263 material. For that matter, you could go back and justify ordinary regression and here’s how you do it, like somebody suggested.

That's actually the set of acceptable ride qualities as judged by horrible long simulations, questionnaires with pilots and blah, blah, blah and whatever, you know, so all right.

This is where people like threw up and stuff over here. This is where they crashed actually. This is where they threw up, but anyway, so what do you do next. Precisely, you just make an ellipsoidal into an approximation. Probably in this case, you might want to make this one. Again, it's you know, you might want to make the inscribed ellipsoid.

Then you take this and you go back to the intern who has only taken 263 and you say, "This is the norm you're going to use. And they, yeah it's a stupid 6 X 6 matrix or 8 X 8 matrix of a bunch of entries, and they go, "Where did all those come from?" And you say, "Last year's intern," right.

It was a lot of work to get it, but you see my point here, that you can actually – by the way, the ideas I'm talking about as far as I know, like no one actually uses them in practice, but everyone should use these ideas in practice, right. I mean this just comes up everywhere. I mean if you're in control, a lot of the methods actually fielded involve quadratic forms like Q and A: and S, then you ask the truth is where do they come from. People make them up.

They don't have to be made up. You can actually do something intelligent and actually get things to do shape the way you really – that do actually capture, not exactly, obviously not exactly. But crudely, they capture the shape of the set.

By the way, over here, this sufficiency was a factor of N. That's something that only matters to a theorist. Okay, and by the way, can anyone guess what's the worst set to ellipsoidally approximate? Just in R2.

Student: Something like –

Instructor (Stephen Boyd):Let's first talk about ellipsoidal approximations in R1. That's a short conversation. How does it go? How well can you approximate a convex set in R1 by an ellipsoid in R1?

Student:[Inaudible].

Instructor (Stephen Boyd):Oh, very well because they're both intervals. Okay, so that was the N equals 1. Let's do N equals 2. Well, you know it can never be worst than 2. Can it be as bad as 2, a factor of 2 – that's if theory says 2. And the answer is sure. A simplex is like the absolute worst thing you could be asked to approximate because in that case, you're outer ellipsoid looks like that. Your inner one looks like that. They're 2 to 1. That's the end, and that's general.

So this N here cannot be made better. Now, a simplex is kind of a weird sick thing. It's got all sorts of corners, you know it's got corners on it. In fact, it's a sort of, it's as pointy as a convex set can be roughly, okay.

Now, most what [inaudible] a lot of convexes that come up in practice. The approximation number is definitely not N. It's often much, much less, and for example, if we actually went out and rode and worked out ride qualities or things like that for a vehicle or anything like that, you would find most sets that arise through natural causes – what I'm about to say is, of course, just total hand-waving, but I believe it to be the case.

Well, actually since I'm not making a statement, it cannot be disproved; however as a rough idea, I will say this, most of the convex sets that come up through natural causes can actually be approximated by ellipsoids stunningly well. Nothing close to square – to N. Okay, so I just mentioned that. That's for those of you who are worried about that factor of N; you needn't be. Okay, so that's our discussion of that.

Centering – that's an interesting thought. So in centering, it works like this. You have a set and you want to find a point that's deep inside the set. By the way, we've already seen one application of this, which is design centering, which is yield maximization.

So in yield maximization, this set describes the set of acceptable values. In other words, the point that you're looking for is what you tell people to shoot for and manufacture. That's what you want to do.

I mean you can also think of less socially positive applications. This could be the range of points where if you're within there, you take out a target. And then you want to ask somebody you know, "Where do you put your sight?" Right, and that's another question. The answer is you don't put it there, and you know, you put it right in the center, what center. You want to maximize the probability now that you're in the set.

But we'll go back to manufacturing, okay. So here, now the simplest, actually the yield [inaudible], we already talked about that, that's actually a convex problem provided the probability distribution is log concave. Doesn't mean you can solve it. You can, but not by methods from this quarter, but you can solve it. Abstractly it's a convex problem.

So a lot of people use a various eurisitc for that. They work unbelievably well. One is this – you find the center of the largest ball that fits inside the set depending on what the set is, I mean if it's a polyhedron, for example, we already looked at that like Day 1 for linear programming.

This is a linear program to calculate the maximum volume, but the largest ball that fits inside the polyhedron. That's an LP. By the way, I can say largest because you would only say largest ball if this were – you'd only use the word largest if there were linear total ordering.

For balls, there is because – at least if you're talking about the size of a ball, it's the radius and so the totally ordered. You would never say what's the largest ellipsoid inside a set because that makes no sense. You have to put in something like largest volume ellipsoid or something like that.

Okay, so that's an LP, and we just worked out this.

Calculating the maximum volume ellipsoid in a set is also a tractable problem and this is call the maximum volume ellipsoid centers. So it's called XMVE. This is X [inaudible], and you can think of lots of others. One, this one has a very important property.

It's affine invariance. That affine invariance means if I transform the whole problem by, for example, scaling the coordinates, if I stretch it some way, that will change radically the [inaudible] center.

But it won't change the maximum volume ellipsoid thing because everything will transform by the fact the determinant of T where T is the linear part of the transformation. So this is affine invariant. Now as to which is better; which is not.

It just means that if someone's calculating in [inaudible] center, you should ask them the following question. How well do you trust your choice in coordinates, you know, have you scaled everything properly? You know, this king of thing. Is it true that X3 being on the order of 1 is about the same as X2 being on the order of 1.

If they don't immediately answer that question with, "Oh, yes, I've been very careful to scale everything here, very carefully so they're all on the same order," they don't immediately answer it that way, then you need to poke them and bug them and say you better check your scaling, because it matters here. Here it doesn't matter at all, those affine invariant.

So, okay. Another center is the so-called analytic center of a set of inequalities [inaudible]. That's a typo. So this we're gonna look at in great detail in two weeks from now so when we actually talk about [inaudible] point methods and how do you solve all these problems. So we'll save that for then mostly.

But it's this. What you do is you set up a problem. You have some equality constraints and you have some inequalities, and these don't really matter. You want the margin here is actually minus FI of X. So for example, if minus FI of X is 0, you'd say it's tight. Minus FI of X is 1, would say you have a slack or margin of 1 and you kind of want to minimize the margin.

And there's lots of ways you could maximize the minimum margin, but an interesting one is to maximize the product of the margins, which is the same as minimizing the negative sum of the logs of the margins, okay. So that's another weapon.

It sounds odd, but we will get to what it means later. We'll also see that although this looks rather complicated, it turns out it's shockingly low complexity, so we're talking 15 lines. Indeed 15 lines that you will write soon. So, and it turns out here by the way in this case, there's also an ellipsoid inequality about if you calculate this you'll get an inner and outer ellipsoid. Yeah.

Student:[Inaudible].

Instructor (Stephen Boyd):Yeah.

Student: How would you extract that center?

Instructor (Stephen Boyd):That's part of the data. I mean I forget how we parametize it with B or D; does anyone remember. Yeah, so that's easy. Yeah, you might have to do some calculations, but whatever it is it's very, very straightforward. You just transform it. I don't know where I was – no, forget it; it's easy enough. It's a quick linear algebra calculation to get the center.

Okay. So here's an example – this will be important later, so I don't mind going over it. Also, by the way, just the idea of an analytic center is something that in many applications should be propagated. You'll see lots about this in the next two or three weeks and related problems, but it's something that should just be propagated because anytime you find a problem where someone says, "Here my specifications. Here are my inequalities. Pick me a point in them."

And you say, "Really, any point? Like what if I picked a point just barely in them," and they go, "No, if you're going to pick a point in them, you might as well get one that satisfies a bunch of them, you know or something like that." Analytic center is actually probably a good choice and it actually comes up in a lots and lots of already comes up in things like maximum [inaudible] estimation and we'll see all those connections; okay.

So percent of linear inequalities, we have a polyhedron, and these are the level curves of this – this is a so-called log barrier function for this thing, this set, and you can see the level curves here. When you get really high curves, they kind of hug the shape of the polyhedron.

At the minimum, it's a smooth convex function. So it's got a minimizer here. That's the analytic center and the analytic center here, this is a smooth convex function near that minimum, this thing looks like a quadratic period. Therefore, the levels sets near the analytic center are ellipsoids.

That ellipsoid is a pretty good approximator of the shape of this set and indeed there's a bound. Now, the bound is interesting. The bound, so you can puff it up by, well a factor of M, that would be M-squared. You could puff up the ellipsoid by a factor of M, but M is not the space dimension, it's the number of inequalities, so it's a worse than the maximum volume ellipsoid or L'owner John ellipsoid. So that's it.

We'll get to these ideas later. Okay. Let's look at another topic is the idea of discrimination classification. It's got lots of names, and let's see how that works and so you want to separate. Here I have a bunch of points in RN, and they're labeled. They have binary labels. They're binary classification, so I can think of it this way.

These, I'm labeling them actually by the symbol, so X is one set; Y is another. This could be a bunch of vectors of something where there actually was something present and this could be a bunch of vectors where something was not present or something like that, but I know which is which here. I'm told which are which.

And what I want to do is this. I mean the oldest problem and by the way, linear discrimination, this goes back easily to MIT in the ‘40s and probably earlier than that, so this has a long, long history of linear discrimination.

So what you really want to do is find out is there a separating hyper plane for this data set, very basic question and if there is, it means is the following set of inequalities feasible, that the X is lying on one side and the Y is on the other. Now, the variables here is A and B.

Now, there is one difference here. These inequalities are strict; okay. In fact, we haven’t dealt with strict inequalities yet, and we’re going to now and you’re going to see a trick. By the way, let’s work out the non-strict classification – let’s have that discussion right now. Under what circumstances can two sets of points in RN be non-strictly separated? That means when does there exist an A and B for which let’s say this is true. When?

Student: Always.

Instructor (Stephen Boyd):Always and by what choice of A and B?

Student:

Zero.

Instructor (Stephen Boyd):That’s right; zero. If I choose A equals 0 and A and B equals 0 and this always works, even if the points are all like messed up and next to each other. In fact, what if they’re identical. So that’s why it doesn’t make any sense to look at that problem. We have to look at the strict problem. You haven’t – we haven’t looked at strict inequalities yet.

So now I’ll just tell you the trick is very, very simple here. These inequalities – take a look at them. They are equivalent to a set of non-strict inequalities. They are equivalent to these. Right, put a 1 and a minus 1 there. Okay. This trick you will be carrying out this trick. You need to get it. Okay.

Here’s the argument. Watch this. If there existed any A and B, where a strict inequality held here, and strict inequality held here, this is homogeneous in A and B. So is that. So suppose the A and B you give me that satisfies this has this. These are, you know, 1E minus 5 and these are minus 1E minus 5. I can multiply those A and B by ten to the 5 and make the gap plus and minus 1. Okay, so I can make it this way.

Now, conversely, if A and B satisfies these inequalities, obviously these imply that. So that's how you do this, so these are tricky and you have to – and if you do these wrong or casually, just you know, normally, there's a lot of cases where it is strict inequality can just replace it with a non-strict inequality, and no one's gonna get hurt and in fact, the original one didn't even make any sense.

Here's an example. If someone says to you, I'm designing a circuit and the power can't be more than 50 milliwatts and you say, "Really," and then later you get a design document and it says as Item No. 1, P is less than 50 milliwatts, like that. It probably means whoever made this up hasn't thought very much or whatever, because you'd say, "Really, could it be exactly 50," anyway.

This is silly. But they probably meant that and there's no engineering difference between the two. Makes absolutely no sense because you'll judge this by spice or something like that and your error will be, you know, it won't be that good anyway.

Anyway if you're manufacture, it's gonna be plus or minus several percent anyway. Just change the room temperature and it's gonna change. So the point is, that's a case where the distinction between strict and non-strict is totally irrelative. This is just due to the ignorance of the person who wrote the specifications. They didn't know there was a difference, okay. So that's this one.

This is not one of those cases because if you just casually make these non –strict here, the answer is 0, okay. If you're using some tool to get the answer, you'll not get an answer, because if you give a problem to a solver like that, it'll be very, very happy to give you the answer and it won't be the one you want.

So there are cases where the strict versus non-strict inequalities actually matter in practice. They matter a great deal. I mean also this all matters conceptually, too. So you should not confuse the two ever frankly, but okay.

Now, to check if there's a set if there is a separating hyper plane is actually just a set of linear inequalities. That's this, okay – in A and B. So for us it's an LP feasibility problem. There's nothing more to say period. Okay, we will say more; we'll say it next week, but that's it.

Let me just mention something that you might use this for. You might use it this way. You might actually take a whole bunch of snapshots of stuff where something actually happened like a target was present or something. Or a disease was present or something. This can be gene expression data.

This could be the gene expression data, giant piles of gene expression data where disease was not present. Okay. These of course, would have a dimension. These Xs could be a million dimensional. For example, something like that. Okay, then what you want, I mean if, then supposed there were a separating hyper plane. There wouldn't be.

But let’s imagine that there were one. If there were a separating hyper plane, you could actually now do something quite interesting. Somebody shows up, you do the gene expression or array. You get the data and you plug it in and this number here will either be positive or it’ll be negative.

If it’s minus 3, you might guess with some confidence, depending on if you believe any of this, that some disease is present. By the way, if it comes out about 0, what would you – that’s a good time to say we don’t know. So we’ll get to the shades business. So that’s the kind of thing you would use separating hyper planes for so it would be to basically predictions of new points where you don’t know which of the two outcomes actually will happen you want to predict. Okay, so we’ll quit here.

[End of Audio]

Duration: 77 minutes

ConvexOptimizationI-Lecture13

Instructor (Stephen Boyd):Okay, so let me start with an announcement. Yesterday's section, which is to say the section which would have been yesterday had yesterday not been a holiday, is actually rescheduled to Wednesday. Now I think it's [inaudible], but of course the website is where the – you'll find something that's more accurate. I was gonna say the truth, but it's merely more accurate. So that'll be tomorrow.

These will be online, the sections, but they're actually only online at the SCPD website, so which I figured by now I'm sure you figured out. And then I wanted to make another announcement that was just sort of general. The TA's and I spend a lot of time like putting together the homework problems, and, of course, we're starting to think about the final, not that you should, but we have to. Anyway, so I just wanted to let people know if you have a – if there's a topic you're interested in and we haven't covered it or something, just let us know.

I mean not that we're gonna change things, but if there's an application area, wireless, signal processing, communications. It's gotta be something everybody can get, but you – where the entire setup takes one paragraph and no more. But if you have – if there's an application area you're interested in and we're not covering it because I guess for some reason, I don't know. We keep doing log optimal finance and I forget what.

We keep falling back on in various maximum likelihood problems or something like that. So, but if there's areas that you're interested in, just grab a TA or me and let us know and maybe we'll do something about it. So, in fact, if any of you have a question, maybe we'll end up putting it on the final or something like that if we can put your research project on the final or something like that.

Okay. We'll continue with our whirlwind tour of applications in – they're geometric. Last time we started looking at this, at the basic linear discrimination problem. By the way, this is very old. This goes into the 40s, this topic. And we ask the question when can a bunch of points labeled? I've labeled them by changing the symbol from X to Y. So I have binary labels on data in our end. And the question is when can they be separated by a hyperplane? This problem goes back to the 40s.

Embarrassingly, for many people who worked on this, it was very strange that, in fact, for years, decades, it persisted. The people did not know this could be solved by linear programming. I mean some did always from like day one. Others did not for a very long time. So that's probably even true today that there are still people who don't know this is the most elementary linear programming.

Anyway, so the question was when can you separate, and we did this trick. This trick you're gonna see a bunch of times. You have strict inequalities here. Now, of course, in general strict inequalities, in many cases, there's no practical distinction between strict and non-strict inequalities, in many cases. In this case, it's – there is a huge difference. If these are non-strict, well, then A = B = 0 works always. So here it's actually – the strict is serious. It changes the problem.

And the trick here is to notice that these are homogeneous inequalities in A and B. And so they can be – these strict inequalities can be replaced by non-strict inequalities, but with a gap, so 1 and -1, okay. So this is a trick. You need to know it because you're gonna see it a bunch of times. It's how to handle strict inequalities that really do matter, strict inequalities in homogeneous problems.

Student: What did you [inaudible]?

Instructor (Stephen Boyd):If they're not homogeneous, well, depend – no, you can't use this trick, that's for sure. So you have to argue specially in that case what to do, I mean in each case, case-by-case basis. It could be that they're not relevant; I mean that the difference is not.

Student:[Inaudible] just –

Instructor (Stephen Boyd):You can do that, but you'd have to argue what you're doing is right, yeah. So that's what you'd have to do. Okay, actually, we'll talk a little bit about when we get to something called phase one programming, methods for determining feasibility. So we'll get to that.

Okay, now you can also – I mean, of course, the set of separating hyperplanes, that is to say the set of AB that separate two sets of data, provided they separate, provided they separate, the set of AB is a convex cone. In fact, it's a open polyhedral cone, okay. So it's a open polyhedral cone, or if you like, down here in this parameterization, it's a closed cone, but this has got the – this is in force to separation. Actually, it's no longer a cone now. It's just a convex set.

So then that means that if – once they separate, you can determine – you can go – you can now optimize something about the hyperplane. Probably the most obvious one is to make a hyperplane that sort of has the greatest distance between it and the other ones or something like that. So one way to say that is to work out the thickest slab that lies between the sets, and the way that works is this. If you have two hyperplanes, for example, well, the set defined by terms Z + B is 1 and then minus 1, the distance between them is 2 over the norm of A, right.

And that in general, A transposed Z + B is the signed distance to the hyperplane A. It's the signed distance provided you'd divide by norm A, 2 norm of A. It's the signed distance to the hyperplane A transposed Z + B = 0, okay. So that's what that is. So the thickness of this is 2 over the norm of A, so that's the – so then if you want to maximize the margin here, that – if you want to maximize this, it's the same as minimizing norm A2 and so you end up minimizing norm – ½ norm A2. The ½ is irrelevant subject to these now inequalities to the 1's and -1's.

So this is – and this is a QP and it's a QP in A and B and it will give you the thickest slab that separates two sets of points, so provided they are separable. Otherwise, this is infeasible if they can't be separated. Now, by the way, geometrically there's something that should be kind of clear here and we'll see that it comes out of duality anyway. It's this. If this is the thickest slab, so this problem is basically put a slab in between the two sets of points and then thicken it until you can't thicken it anymore.

Well, it turns out that is – turns out that's exactly the same as the following. You calculate the convex hull of the two categories of points like this, like that. And you calculate the minimum distance between these two convex hulls, okay. Actually, that's an upper bound on this maximum, and, in fact, it's equal to – Jim, I'm just – this is just arguing intuitively. The thickness of the maximum slab – so finding the thickest slab between two sets of points, that fits between two sets of points, number 1 and number 2, finding the smallest distance between the convex hulls of the points.

Geometrically, it looks like those should be duals. They should have equal value and so on. And, of course, that pops straight out of duality. So roughly speaking, it says that separation problems, maximum separation problems, thickness – you know, when you want to maximize, the thicknesses, the duals of these typically are distance problems, distances between convex hulls, oh, and vice-versa. So what this means is when you're solving, for example, a minimum distance problem, the dual will always have an interpretation of somehow putting a fixed slab in between things and vice-versa.

Okay, let's work out how that works. Well, we'll just go back to this problem, sorry, the QP here. So we go to this problem here, max origin problem, and we're gonna work out the dual of this problem. And I'll just – I won't go through the details now because you can do this on your own. You end up with a a Lagrange multiplier, lambda and mu.

These are the Lagrange multiplier for these inequalities and these, these lambda and mu for these. And you end up with something that looks like this. You want to maximize one transposed lambda plus one transposed mu, some of the lambdas plus some of the mus, subject to this second order cone inequality here, a single one. Then the sum should be equal and they should be positive, okay.

Now the optimal value of this is equal to the optimal value of this problem. All right. So here I'll change variables to theta I divided by 1 transposed theta. What I have to look at, you have to realize here that lots of things here are homogeneous here. So if I change to theta I's, which is the normalized lambdas and the normalized lambdas and then gamma I's are normalized mu's. You end up – and then you take T as this variable here. You invert the objective, so instead of that, we'll minimize 1 over the objective. We'll call that T. Then you end up with minimized T subject to this. And if you'll look at this carefully, you'll see exactly what it is.

These two say that theta is a probability distribution or a set of weights. Therefore, that is a general point in the convex hull of the XI's. That's a general point in the convex hull for the YI's. This is the difference of the two of them in norm less than T. We're minimizing T, so that's the same as minimizing this thing. So it's a distance between the two convex hulls problem, so that's what you get here. So again, I didn't – I don't expect you to be able to follow any of this. Just the idea is just to be able to see that they are related by duality.

Okay, now mention a couple of things, some of which are, well, both highly hyped and highly useful. We'll get to some of that in a minute. So we can talk about approximate linear separation of non-separable sets. So actually, in many applications, that's quite useful. And if you want to put an application in your mind, let's let me give you one. You take email and you take a training set and you flag it as spam or not. In other words, a person flags it. It just said, "That's spam. That's not." And you take the email and you calculate various features.

A feature could be anything, so that's what an expert in this kind of thing would do is go and work out what the features would be. It might be something – it doesn't even matter what the feature is. It might be via currents or not of a term. It might be the frequency of a current. It might be the distance between two terms. It could be all sorts of weird stuff in it, but an expert will go analyze the text, not just text, but also technical things like maybe where it came from and all that kind of stuff.

And you'll make a feature vector, and the feature vector might be, let's say, 10,000 long. So each – remember back to exactly this thing. So these are now vectors in our 10,000 or something, or roughly. It doesn't matter, 1,000. That number doesn't matter. It's not 5. There's lot of – you have lots of features here, okay.

And what you're hoping is that, in fact, there'd be a hyperplane that would separate the spam from the non-spam. I mean that would kind of – and then it's great because a new email comes in. You calculate the features rapidly and simply find out what side of the hyperplane you're on. And that's your prediction as to whether it's spam.

So everybody, this is the application, okay. So, in fact, you're not gonna get a separating hyperplane. It's essentially impossible. So what's gonna happen is a couple are gonna be over here, right, that you're just gonna miss on some. Real data sets are gonna have a couple of points here and a couple of points here. You still want to deal with it. You want to come up with something that, I mean, there's lots of ways you can imagine approximate separation.

The simplest would be this. Okay, the most natural would be this. I give you two sets of data and actually you verify it by solving the linear discrimination problem, and in fact, they are not separable by hyperplane. That's fine. Then what you want to do is this, and you might ask, please find me the hyperplane that minimizes the number of missed classifications. I mean that's just totally obvious that that would be the kind of thing you'd want. It makes perfect sense, right? You'd say, "I have, you know, 30,000 emails, and if, in fact, I misclassify .1 percent, for example, that's still a bunch of them."

That would probably be okay. That would probably be quite good, so that's – but it'd be fine. The idea that you'd get all 30,000 exactly right is basically generally not possible. Yeah.

Student: Do you [inaudible] the size of an airplane?

Instructor (Stephen Boyd):Oh, absolutely. We're gonna get to that momentarily, yeah. So you could do – yeah. Actually, one of the big questions or big pictures or conclusion is it turns out things you can do in a low dimension space with all sorts of curved separators and things like that, it turns out a very good way to do those is to blow them up into a huge dimensional space and do linear separation there. So that's why you hear a lot of people talking about focusing a lot on linear separation. It's generally linear separation in a much larger space, right, which actually could involve non-linear stuff.

For example, one of the features in this space could be something like the product of the number of occurrence of one word and another. Given our application, I won't say those words, especially because we're being taped right now. But, for example, pairs of – so in that case, you actually – you think you're doing something that's linear in a high dimension space, but, in fact, you're really doing something non-linear because some of the features are products of others or whatever, okay.

I don't know if I answered that question because it got less clear actually. Anyway, let's move on. So you want to minimize the number of misclassifications. Well, it turns out that's impossible. That's gonna be hard to do. Actually, you know what? I'm not entirely certain of that, so let's just say I believe that's hard, that's a hard problem, so is minimizing the number of misclassifications. However, we have a very good eristic, which you know now, and it's based on these L1 type ideas.

So you know that, for example, if you minimize an L1 norm of something, you're typically gonna get something that's got – that's sparse. And so this is an exceedingly good eristic for minimizing cardinality, which is a number of non-zero elements. So here's what you do. I'm gonna put in a little fudge factor that's UI.

Now, if UI is – these are gonna be positive, by the way. The UNV are positive. If I put in a – if U is 0, if UI is 0, it means that inequality is satisfied. That means everything's cool. You're on the right side of the hyperplane. You're directly classified. But I'll put in fudge factors and they'll be positive, non-negative. If they're 0, it means no fudge factor was needed for that point being on the right side of the – actually, not hyperplane, but slab. That's right. It's on the right side of the slab if UI is 0, or if VI is 0, it's on the right side of the slab. Everybody got it?

Now you – if you minimize the sum of those what you'd really like to do perhaps is minimize, in fact, something like this, the number of non 0 used in these because any time you threw in a fudge factor, it means you failed to meet that – to classify that one correctly. You failed to classify that one correctly and you need a fudge factor to help you classify it or whatever. So if you minimize the sum like this, and by the way, these could be weighted, but this is gonna be a very good eristic for linear separation of non-separable sets.

And indeed, if you solve this problem, it does really well, and you get something that will typically misclassify a very small – a small number of points. Is it the minimum number of points? No, we don't know and probably don't know. By the way, I don't know what this is called the machine learning, this particular one. Does this have like a – usually since they imagine they invented everything they would have a name for – somebody in machine learning, is there a name for this one?

Student: Support detriments.

Instructor (Stephen Boyd):And isn't this the limit of the support vector machine at one end or something? They probably – I would imagine they'd make up some fancy name for it and then write multiple books on it, but we'll get to that momentarily. It's just this, okay. This is the support vector machine, but it's a limiting case of one, okay. So this is an example where the sets are not separable and you can see it's done quite a good job of doing this. Of course, all these pictures are silly because the applications of these, I don't have to tell you, are not for separating clouds of points in R2 because your eyeball is excellent at that, okay.

This is for distinguishing binary tagged data in dimensions like 10,000 and up where your eyeball is not so great. So that's what this is for. Okay, this leads us to the support vector classifier, and here there's actually – you make it a buy criterion problem. And the buy criterion problem is this. You want to – this is sort of the fudge factor here. But in fact, here you can also trade off the width of this slab, and if you do that, you end up with something like this. This is one version of what's called support vector. Oh, and this is just so pedantic. It's called SVM by people in machine learning and things. Anyways, remarkable.

And so, I mean, it's nothing. It's just two things. This is basically the inverse width of – it's two times the inverse width of the separating slab. This is an extremely well known, totally elementary eristic for making this sparse, meaning please misclassify few. And you get this, which is nothing but a QP, all right. That didn't stop people from writing like multiple books, all of which are more or less incomprehensible. Many of them are completely incomprehensible, but anyway, so it's called support vector machine.

Don't you like that? A machine, a support vector machine. It's great, isn't it. It's kind of what – okay, so it makes it sound fancy and so on. Okay, so that's it, and here is, I think, the same data set. It is indeed, but here's the same data set. Here we put – let's see what we did. We put less weight on gamma here and what that meant is we spent more effort making a small. You make a small, 2 over norm A is the width, so this thing opened up like that and I got this one. Actually, it's a bit different. Let's see, I don't know if I can overlay them, but well, I can't see it, but maybe you can if I go from there to there.

Yeah, it twisted around a little bit or something like that, but okay. Now, even though it's sort of hyped, I have to say the following. These things work unbelievable well. So, in fact, things like spam filters and things like that, of course, you have to have the right features, but generally speaking, these things work like really, really well, which kind of make sense. Things that are over hyped and then don't deliver kind of go away. This is not going away, so I assure you of that, so okay.

Somebody asked about linear versus non-linear discrimination. You can do anything. I mean you can easily separate things by non-linear function. For example, you might linearly – you might have a function which is itself a – you would consider a linear – a subspace, in fact, of functions. Now that's exactly what we did here. So far we do linear classification, which you might really call affine classification. The class of functions you look at are affine.

Here they could be anything you like. It wouldn't really matter. It doesn't – what matters is the following, is that the function family that you look at is linear in the parameters. That's all that matters, in fact. Then everything works. You can do discrimination with anything you like. They could be polynomials. They could be all sorts of crazy things. Well, I mean, for example, you could make them gaucians and things like that and you could do discrimination that way, and gaucians parameterized with different widths and different positions. And you could get quite fancy and do separation that way.

But as far as we're concerned, they're all very simple. The base problem is always a linear program, so period. And just as an example to see what you can do, suppose you said quadratic discrimination. That basically says you parameterize F by P, Q, and R. And so you're searching a solution of these inequalities. Now the variables here are P, Q, and R, so, in fact, these inequalities in P, Q, and R are what? They're what? What kind of inequality is that in P, Q, and R?

Student: It's linear.

Instructor (Stephen Boyd):It's linear. It's just linear. So this is nothing but a set of linear inequalities in P, Q, and R here, but obviously now I can add other constraints. If I make P positive, definite, or something like that, then, in fact, it means that this one will be – well, negative definite. It means that this one is actually an ellipsoid. So I can add the constraint. You know, please make me an ellipsoid that covers my data and cover – and doesn't contain as many of these.

And, of course, once you get the basic idea, we could even make a – let's see, how would you call that? A support ellipsoid machine, SEM, the new rage in machine learning. You can – so you can say, "Well, what if you find me an ellipsoid that trades off, so size of the ellipsoid versus number of misclassified or their blah, blah, blah." And we could easily make this up this way. It'd be extremely simple to do, right, to do that. And then you'd get approximate ellipsoidal separation and things like that. But let's just try a few – I'll try a few things on you and see what you think.

What if I said suppose the dimension is huge and I only want to look at, for example, P’s that are banded with bandwidth 5? And I – will that work? Did I solve separation by a P, which is banded with a width of 5? Sure, yeah, it’s just a subspace. It’s no problem, right. So those kinds of things will work, okay.

Here’s another example where there’s a bunch of data here and this is separation by a fourth degree polynomial. For example, there’s no third degree polynomial that separates them. So you could solve by quasi convex optimization, clearly the problem if someone says, “Do you find the minimum degree polynomial to separate these sets?” Actually, more generally, if I gave you any – if I set a basis elements, but ordered, so gave you an ordered set of basis elements like F1, F2, F3, and so on, and then I said, “Find the minimum number of these taken in order to separate the points.”

That’s quasi convex because you just do – you solve an LP feasibility problem with some number of them and then you know whether you can separate with that number of basis elements or not, okay. So that’s that, that’s the idea.

Okay, we’ll move on and look at one last sort of topic in a family, problems in geometric optimization problems. And these are placement and facility locations. Again, there’s tons of varieties on these things. There’s whole books on these, so the idea here is just to give you the flavor of that.

So placement and facility location problems generally look like this. You have a bunch of points or coordinates. Now often, these are in R2. Sometimes they’re in R3. And they’re – but they’re often not in higher dimensions, but there’s no reason they couldn’t be. So they could be in R2 or they could be in R3 or they could be in R4, which would be something like space time or something like that.

Then what happens is this. You have a bunch of points. We’ll just make it R2. A bunch of points are given and others are variables. So, for example, what happens is you’re given – these are – well, I’ll make these fixed, and then the variables are these. So you can think of these. There’s many names for these. These are sometimes called anchors, depending on the application because they’re the ones that are fixed and don’t move, and then these are sort of the free points. They move around, okay. Oh, the other name for these in circuits is pins, for example, or something like that, fixed cells or pins or something like that.

And what happens is for each pair of points here, you have a cost function that is a function of the pair of distance. In fact, usually it’s a function of a norm of the difference, but it could just be a function of the pair. And I’ll give you some examples here. Let’s do circuits, for example. So in circuits we’re placing cells here and what I will do is this. These are fixed. There are maybe pin – these are external pins or other blocks or whatever that you’re gonna – that are just – whose position is fixed. You’re doing placement.

And these are cells that can move around, and I have a net list, meaning I have connections in between a bunch of these things that look like this, something like that. And, in fact, each of these might have a weight on the edge, which would be the number of wires that connect one cell to another, okay. And the absence of an edge means there's no wires connecting it or something like that. So, for example, there's no wires directly connecting this cell and this one, for example.

And now your goal is to position these to minimize, for example, the total wire length. That's a very common problem. And, in fact, the wire length would typically be done – I mean if you care about this, would be done in a L1 norm because you do vertical, horizontal routing of wires in a circuit, for example. And there's all sorts of things that make this more complicated and so on. This could be sort of a radio relay network or something like that and the distance between two nodes could be the following. I'm sorry, the cost could be the following.

It could be the power required to establish that wireless link, right? And that would be, for example, a function of the distance, for example. And in that case, I'd say, "Please move my mobile forwarding stations in such a way that the total power used by the network is minimized, for example." I mean I'm just making up examples. They just go on and on and on like this. Or you could say, "I have a distribution of a commodity. These are the warehouses. Each of these lengths could have a – each of these could have a weight, which is the amount of traffic that goes between these two places, and you could ask to put this in such a way that the total amount of freight times miles is minimized, just for example.

But okay, so that's a typical placement problem. Okay, now these, of course, are all convex, these. By the way, the minute you start doing placement, there's all sorts of other stuff that immediately pushes it out of the convex realm, and I'll just mention a few of those just for fun. One is if you're doing placement in circuits. There's another rather important constraint. That is that cells should not sit on top of each other, so that's not an overlapped constraint. That's not convex by any, not even remotely because you're placing cells. So that's one problem.

And, of course, in all the other ones, they're silly. It turns out, for example, if you have to go – if you want to look at the power required to transmit from two points, it might not be just a function of the distance. It might depend on the terrain and things like that. Then this gets very – then it's clearly, again, non-convex and so on. But first cuts are typically convex like this.

So here's just a simple example. So we're gonna minimize a function of 2 norm of the distance of the pairs. We have six free points, and I think the anchor points are these around here, and there's 27 lengths here, and here are the six free points. So if you minimize the sum of the norms, you get this picture here. If you minimize the sum of the squares of the norms, you get this. And you can actually see what's happened here.

Oh, by the way, you're not gonna get stupid with these things. You're not gonna get stupid placements, right? For example, this node is not gonna be – no one is gonna place themselves outside the convex hull of the anchors, for example, right? Because moving it just to the boundary will decrease this. You're not gonna get this point sort of way over there, right.

Oh, I should mention one other application of these things. One is just drawing or visualization, so that's a perfectly good example. So, for example, I might have drawing your visualization would go like – you know, I'd have a bunch of points. And I would have some measure of sort of a similarity. And you want similar things. You want to plot this on a screen or on a piece of paper, and you want similar things to be close to each other. So what I have is a dissimilarity measure or something like that.

And so I want to place the points of the similar things close and dissimilar things are far away, just for example. And this is the kind of thing that would do that. It would cluster similar things and so on. Okay, so here you can see what's happened is here you have a lot of – in fact, I should mention something here. If you minimize the sum of these norms, that is just like an L1 type thing. So what's gonna happen is a bunch of these are gonna be equal.

So if I have six free points, I think – let's see. Did I actually have – one, two, three, four, five, six? Okay, so I believe, but I can't see closely enough here. My guess is that two of those, at least one point is sitting exactly on top of another. They cluster. So if you minimize sums of norms, they will cluster. It's just like an L1 type problem. They'll cluster here, okay.

And you will have some long ones. And if you look at the histogram of interconnection lengths, you'll see you have some wires that are as long as or lengths that are as long as 1.5 and so on. And you have – let's see. Well, so we may have one that's 0. There's at least one that's very small, might be actually exactly 1. My guess is it's exactly 0. In the – when you go to squares, what happens is these long lengths irritate you.

Well, they irritate you squared compared to just linear irritation, and so this spreads out so that the longer lengths are shorter, which is better, and you have bigger distances in the middle like this. By the way, the name for this is quadratic placement. This is called quadratic placement and used in a couple of different fields. And then you could do something like you'd have the fourth power and that will spread it out even more.

Let me ask you a couple questions here. We'll do it this way. What if H was this? You get a free ride from 0 to 0.3 and then it grows linearly. What do you think the placement's gonna look like?

Student: It's gonna be a lot of 0.3.

Instructor (Stephen Boyd):Yeah. It says basically distance is free up to .3. So what's gonna happen is when you solve this they're actually gonna spread apart. And a lot of them are gonna be just exactly .3 away from each other, right? Some of them will be longer because they may have to be. Some will be shorter because it doesn't matter, but a lot of them will be about .3 away from each other, so that's what will happen here.

So, all the same ideas once you start understanding about how to construct these functions. Everything comes into play. I mean I could even do this. I could then make this go to plus infinity at .5, and then, of course, no wire length will be less than .5. It might be infeasible, but if possible, it will arrange it no wire length is less than – if there are wires, no wire length is more than .5, for example.

Okay, that finishes up this whirlwind tour. You should be reading the book as well because it goes into much more detail on these and the variations and things like that because, first of all, I'm going fast, and so anyway, you should be looking at the book.

Okay, we're actually gonna start a – I mean that actually sort of finishes one section of the class, although it's not really because it'll just go on forever now. That's our whirlwind tour of applications and we're just gonna – we'll just be doing applications, as I said, from now on, no exceptions, until and through the final. We'll just do lots of applications.

But we're gonna start a new section of the course, and it's actually on the algorithms, so how do you solve these things, about how, I don't know, the stuff, the codes you've been using. How do they work? What do they do? And we'll go through all of that. You'll end up knowing not stuff at the very – at the ultra high end, but actually well enough to implement that work like shockingly well. And then some applications work better than sort of generic custom.

So we're also gonna cover – actually, the stuff we're gonna work on – we're gonna start today is extremely useful. Basically for anybody taking this class should know the material here. Probably some people do. Actually, how many people have actually had a class on like numeric or linear algebra and some of the things like that? So just a handful, okay.

So everyone should know this material, just period, end of story. It doesn't matter if you're the most committed theorist, makes absolutely no difference. There's no excuse not to know this material. So that's my view of it.

And actually it's just a few things will get you very far. That's the problem. If you get books or take a whole course on it, it's just too much stuff in too much detail. And it's so much stuff and so much detail that by the end you've been so pounded with all this material that in the end you can't even remember what the main points were. So we'll do it so lightly and at such a high level that you'll have no option but to realize what the main points are because we're not going into any others, and we'll barely go into them.

So, okay, so this is our – and let me also just put this in perspective. You will know – actually, won't take very long. You will know how to, for example, solve a second [inaudible] program or an STP. I don't, a couple of weeks, that's all. You'll implement one absolutely from scratch and it'll work and it'll work actually like really well, okay. So and it's not a big deal. We're talking like 30 lines of code. It's just nothing. It will be competitive with something that's got thousands and thousands of person hours of development in it, but you'd be shocked at how well it's gonna work.

So let me say a little bit about how it works. Oh, by the way, I should say my position also on all this material. So when I talk to a lot of people, so a lot of people who use optimization are sort of very naïve about it. Everyone is fascinated by this. It's just all the [inaudible]. That's all they care about. So you mentioned something and I'll go somewhere. It's above noxious, but I'll say something like, "That's an SOCP." And they'd say, "Oh, yeah. Oh, great, sure. What's that?"

But then immediately focus in on like how do you solve an SOCP, in which case I'll – and I think sort of the correct response to how do you solve an SOCP is it's none of your business. It's technology. It's just not your business. SOCP's can be solved very well. There's a thriving community of people who do it. There's lot of algorithms worked out, tons of papers on it, open source software. There's also commercial software if you prefer to pay for it.

I mean, so it's sort of not – but everyone just wants to worry about how you do it. Now, of course, if you solve giant problems, that might be necessary, obviously, right. If you solve – so this is why I don't recommend you go into either machine learning or image processing or medical imagine. These are just – if you're already in one of these fields, sorry. It's too late because that means you are gonna have to know how to write your own things because you can't just use generic stuff to solve problems with 10 million variables.

If you're in another field, good, stay there, with a modest number of variables like 10,000, maybe a couple hundred thousand, something like that. And also, avoid fields with real-time constraints. So any field where someone can say to you, "That's great, but that's way not fast enough." Then again, you should avoid that field. Again, this is only if you have not chosen a field to go into, but all right. Where am I going with this? I have no idea.

Okay, no, I just remembered where I was going, yes, yes, of course. Oh, so I want to say that – basically what I want to say is this whole section of the class on numerical stuff is good to know it, but remember, most people will focus on this. If you take a class like this at some other institutions that I won't name. Well, I would name it, but I'm not going to. You will find the entire class, basically from the beginning to the end, is sort of on algorithms. I mean this is weird.

This would be like taking entire class on like TCPIP, right? I mean you can do it and somebody has to do it, right, otherwise the rest of us can't use it. But there's just something weird about thinking that's what's important to teach about this material. What's important to teach about it is actually the applications and the modeling and all that kind of stuff. This is way secondary. But how do you say we're gonna now launch into it?

And let me say what he big picture. So the big picture is this. Let's say you're gonna solve – we're gonna solve non-linear constrained, inequality constrained problems, okay. We're gonna build our way there. And the way that's gonna work is this. We're gonna build it up by each – we're gonna solve each more sophisticated layer by solving a sequence of ones at the next layer. There are just – I'm gonna give you the big picture right now. This is how it's gonna work.

You're gonna – so let's just take an LP, okay. So let's just take – here's an LP. And the way we're gonna solve an LP is you're gonna solve a sequence of quadratic – sorry, of non-linear, but smooth minimization problems. That's what you're gonna do. That's how we're gonna solve an LP is you're gonna solve 20 non-linear, smooth, non-linear minimization problems with no inequality constraints. That's what you're gonna do, okay.

How are you gonna solve these? You're gonna solve each of these by solving a sequence of, in fact, quadratic minimization problems, okay. That's how you're gonna solve these. This is gonna be Newton's Method, by the way. The name for this is an interior point method. Then this is Newton's Method that converts solving a – minimizing a general non-linear convex function to one where you solve a quadratic minimization.

Now, by the way, quadratic minimization is very interesting because if I ask you to minimize a complex quadratic subject to the quality constraint, the solution involves solving linear equations. So this is basically – that's a sequence of linear equations, okay. So what it basically says is the following. When you solve an LP, yeah, you'll do about 20 of these or something like that, and it doesn't real – let's see. Is that about right? No, even less, sorry.

You'll do like ten of these, and in each of these you'll do five of these. And so the point is that what'll really happen is if someone profiles your LP software code, it will discover that it's doing basically only thing. It's solving linear equations and it's doing so 50 times. I'm just giving you the big picture. But I'm going to say that the whole foundation rests on linear equations.

Oh, by the way, we could go lower because linear equations rest on multiplication and this sort of stuff, and you can go down, and that – then we can actually talk about floating point calculations and blah, blah, blah. So you get – I mean we could keep going, but we'll just stop right here. So okay, so this is why solving linear equations is important, so all right.

So we'll start by talking about sort of matrix structure and algorithm complexity. So how does this work? We'll just look at just AX = B with a square matrix, okay. Now, for general methods, the cost of solving AX = B grows like N cubed. And that's just a – that' approximate for some reasons I'll say in a minute. And just for fun, I mean, last night I typed into my laptop, and so how long?

By the way, N cubed when N is 1,000 is not a small number. It's 10 to the 9. Roughly, it's on the order 10 to the 9. Floating point operations go down. How long do you think it took for me to solve AX = B? This is on my laptop, which is two years old last night.

Student:A few seconds.

Instructor (Stephen Boyd):What?

Student:A few seconds.

Instructor (Stephen Boyd):Okay, that's a good number. Any other guesses? Yeah, I mean the actual order of magnitude, you're in the ballpark. That's a lot. To write 1,000 cubes is a lot, okay. So the answer is half a second, okay. So that means on a modern machine right now it would be much – it would be substantially less, but okay. So, I mean, I – you want to get some rough – I mean that's actually why all of this is possible is because basically computers right now are way, way fast. I mean amazingly fast, okay. So I said about half a second.

Oh, by the way, for fun I went to three – I went from 1,000 to 3,000. It should have gone up by a factor 27. Is that right? No, 8? Sorry, three, I did three. I went 1,000 to 3,000 for fun. Should have gone up by a factor of 27. It actually went up only by a factor of 20, but roughly that was it. And I dropped it down to 100.

By the way, what's the prediction? If N = 1,000 took .5 seconds on my laptop, how about N = 100? By the way, the number you're about to work out, it's a very important number actually, and it's something that is not appreciated at all, I mean like throughout the world. So what's the number? I mean if the scale's like N cubed, what's the number?

Student:[Inaudible].

Instructor (Stephen Boyd):Yeah. It's 500 microseconds, okay. Now just to – if it's fine, I think we should probably have like a moment of silence to reflect on what this means. This is unbelievable what this means. That solving a set of 100 linear equations with 100 variables, just the data, there's 10,000 data points. Did I get that right? I mean that's just to describe the matrix, okay. This is not a small think, okay. That goes down in 500 microseconds. That's predicted. And the actual number, by the way, was about, I guess, a millisecond or something, okay.

So these numbers are just amazing. And this is just – this is with all the stupid mat lab overhead and all that kind of stuff. So this is – a mat lab, of course, has nothing whatsoever to do with the actual algorithms that we run. The algorithms actually run in mat lab. They do all the numerics all open source public domains called LA Pac. So don't ever confuse the two.

Okay, so these are just amazing numbers, but they are. So it's just worth – I mean I know everyone knows these things or actually, I know most people don't, by the way. If I go around and hang out with other professors who do stuff that involves like – you'd be shocked at the numbers people would guess like this. I mean well respected people I won't name. I've had people guess recently things like minutes for this, so minutes for a thousand, right.

So, of course, it's not their fault. If you kind of don't pay any attention and a decade goes by, Moore's law propagated by one decade, you know, I'm sure I've done that too. I could say, "Oh, boy. You could probably do that in about a minute." And anyway, you can say that in a couple of classes and you're wrong pretty quickly.

So okay, all right. Back to this. Okay, that's [inaudible]. Now here's the part that's important. I mean this is basically – you should get a rough idea. Everyone should know these numbers. There's no excuse not to. It's silly. Now here's the really cool part and here's actually what we're gonna cover in the next lecture, and this is very important to know. It makes these numbers even more shocking, way more shocking.

If A is structured like banded sparse couplets and we're gonna see a lot of other types, some that people would not recognize, most people, okay. In that case, it's way, way faster. Now, for example, let's just do a quick example. Suppose A is tridiagonal. Anybody know? Someone who's done numerical computing will know how fast you can solve a tridiagonal system. How fast?

Student: It's institution.

Instructor (Stephen Boyd):It's actually N.

Student: Oh, N.

Instructor (Stephen Boyd):It's N if it's banded, it's N. So what that means is, for example, solving a million equations with a million unknowns with a banded matrix, it's like that, okay. I mean these are not secrets or anything, okay. The people who do this all the time know it, actually, but other people should know it. You should know it. There's no reason for you not to know this kind of stuff, okay.

Actually, what will really matter is then we'll make all the neural connections between the application and these things and then you'll look at something like an optimal control problem or signal processing. You'll look at it. All the neural connections will go through and you can say, "That problem I can solve with a million variables, that signal processing problem. That image restoration problem, I can solve that with a lot of variables because blah, blah, blah banded or such and such." So we'll get to all these things.

Okay, so how do you estimate the – an algorithm for solving AX = B? Well, this is really from the 60s, so it's a bit sort of behind, but it's still a useful concept, although you shouldn't take it too carefully. It's a flop count. So flop count is something like a floating point operation. It means it's something like this. It depends on how you do the accounting, and, in fact, it doesn't really matter anymore because it's not that accurate. But it basically means something like an addition, subtraction, right, something like an addition, a subtraction, a multiplication. And it also – something they throw in something like they amortize a little bit of indexing or something like that. So it's just a very crude number.

By the way, it was not a crude number in the 60s and 70s and 80s, maybe early 80s. It was not a crude number because a floating point operation would actually often be like shipped off to a separate chip or something like that. I mean it was like a big deal to do a floating point operation. So counting floating point operations then made a lot of sense and actually would give you a pretty accurate idea of how long something would run.

These days, of course, it's really close to meaningless. I mean it's kind of silly. What matters now, obviously, is things like cache misses, locality of reference in memory. Things are deeply pipelined. All sorts of stuff is going down at the same time. So it – floating point, I mean, you can write down two algorithms, analyze the number of floating point operations, and they can run, let's say, 1,000 times off in speed.

This, by the way, hints that floating point – counting floating point operations is not such a super good measure anymore, that they could be off by a factor of 1,000. Still, it's roughly useful, but it means don't take a number seriously. If someone walks up to you on the street and says, "I have a new algorithm, and it's – you know, the usual one is like 1/3 N cubed. Mine is like .2 N cubed." Then you can just laugh and walk away or something because it's, I mean, so far off below the threshold of significance, it doesn't matter.

Okay, so here's what people do. Again, it's from the 60s and it's not quite right, but it's still a useful idea. What you do is you count up roughly the number of operations required to do something like solve AX = B. And that will generally, in the problems we're interested in, it will be a polynomial function of the problem dimensions. You may be estimating stuff, that's fine. And you simplify by keeping only the leading terms.

So you'd say, "That's an N cubed algorithm, or that's an N to the 2.5 algorithm, or it's fan, or linear, it would be – it's just order M." Okay, so as I said, this is not an accurate predictor of computation on modern computers, although, by the way, that's what I was doing here. Later, I'll actually tell you exactly what was happening in these two cases. And actually, the fact that I was predicting within a factor of two or three is pretty good, but actually only because it was all optimized, but we'll get to that later. But it's still useful as a rough measure, estimate of complexity.

Okay, so some of this is pretty basic, and I should mention something about this. I think everyone should know about it. We're not gonna get into this, but it's just to give you a glimpse of below where we're gonna go in the class at the lower level stuff. Actually, all of this, the linear algebra we're talking about now is done in something call LA Pac, which is linear algebra package which is open source software.

It's among the most debugged software anywhere in the world because it's used by lots and lots and lots of smart people. It's written by very smart people, totally open source, and it's basically what – it's everything you do if you use Mat Lab, Octave, anything else, it's gonna use LA Pac. We'll get into some others, so, and I presume that's true of R as well, but I don't. Do you know? Yeah, there we go. So it's R as well.

Okay, so this is what – and basically below LA Pac is something lower called blast. Again, this is just – you know, it's not part of this class, but it's just if you want to look below the level we're gonna go, this is – these things are very good to know about. Blast is basic linear algebra subroutines, okay. And they distinguish blast level one, two, and three. So blast one, blast two, blast three. You'll hear things like this.

If you don't hear things like this, you're probably not hanging out with the right people, by the way, just let – so if you hear – if most of your friends are talking about squared in, login, complexity, you should at least get some friends that talk about these things just to kind of get a more balanced view of the world.

Okay, so blast level one is generally like things like vector vector operations and things like that. So these are separated logically this way. So these are actually blast level one. So here this would be things like calculating inner product. Well, how many flops does that take? Well, you have to multiply the pairs and then add them all up, and it's 2N – 1 and people just say, you know, 2N or they just say N. I mean, and by the way, some people count flops a factor of two off. So some people say a flop is actually kind of a multiplication and an addition plus a little bit of indexing.

So basically the usage of the word flop is inconsistent by a factor of two immediately, which is fine because it doesn't predict anything more than a factor of two anyway now, nowadays. So you would just say this is order. You would just say this is N. If you add two vectors, I mean, this is silly. It's N flops. What you want to start thinking about here is what happens when there's structure. There's not much structure you can have in a vector.

Well, there is one. You can have a sparse vector. If you had a sparse vector, then how would you calculate the inner product between two? Suppose my vectors are each 100 million long, but each of them only has 1,000 non-zeros stored in some reasonable data structure, right. I mean let's just make this simple. We're not gonna do an exotic one. How would you calculate the inner product between two?

You wouldn't ask each one in turn, "Please can I have XI? Please can I have YI? Multiply the numbers, and then plus equals that onto your accumulator or whatever, right?" You would not do that. What would you do?

Student: Fix it where I'd [inaudible].

Instructor (Stephen Boyd):Yeah. You find the common ones. So basically it would be boising fast. But as a sample, I mean a stupid, simple example of where structure comes to make not a small – we're not talking about small increases here, right. We're talking about dramatic increases in efficiency, right.

Student: How does knowing something's structure help you in actually doing these computations? Doesn't it take some work to actually figure out what's zero?

Instructor (Stephen Boyd):Right, that's an excellent question. Okay, and we're gonna get to exactly that question, excellent. If I don't answer it today sensibly, I'll answer it on Thursday. The question was, you know, how does knowing something's structure help. I'll tell you the answer right now roughly. It may not make much sense. It's this. Some structure will – is actually – can be, in some cases, automatically handled for you. Some types of sparcity patterns and things like that will be automatically recognized, depending on the software system you're using.

If you're writing sort of high level stuff, you know, like in Python or Mat Lab, some structure will be recognized. Other structure won't be, okay. And that's the one that will be the interesting part. If you're writing actually production code or stuff that actually works for something large, it – knowing the structure is there helps you this way because now guess who's gonna write the code for it? You. So we'll get to that.

Now the low level stuff, you don't have to do. A lot of the structures though will already be things they can do. That's just – sparcity is not one of them. There's no absolute standard now, but for a lot of other things there will be. There's actually code low level, so you should know about it. All right, let's move on.

Matrix vector multiply. Here if A is dense or you treat it as dense, then you want to do a matrix vector multiply. You just work out how many. It's basically 2MN flops because you're basically calculating interproduct with each row of an index, and that's what it is. It's that. If A is sparse, if a matrix is sparse, and it's got a round capital N non-zero elements, then I mean that's all you're gonna have to multiply by here because you're gonna just calculate those and that gives you this.

Now if A is given a different data structure, like, for example, suppose A is given as a low rank factorization. Suppose A is – it doesn't matter. A could be like a million by million, but I'd give it as a rank 10 factorization. Note, that would allow me to store A because there's no way you're storing a million by million dense matrix anyway. You can't do it.

But I could store A as two million by ten matrices. That's nothing. And here the key is not – is to store it like this and to multiply A by X by first calculating B transpose X, which gives you ten numbers, and then forming U times X, which gives you million. And that's fast. That's gonna be 2P times N. In this case, if that was a million, this would be something on the order of 10, 20 million flops, which, as you know now, goes down way, way fast as opposed to something that just wouldn't work even remotely.

By the way, to go back to your question about knowing structure and so on, there is no system in the world that will recognize for you low rank structure. So if you, anywhere in your code, write something equals U times B like this, there's nothing, there's no system right now that would check and say, "Silly guy, that's actually rank 10." Anyway, it wouldn't work anyway here because if you formed U times V where there are a million by million, it's all over anyway. It's just nothing. It'll stop when there's no more virtual memory left or something like – it'll just stop at some point.

So low rank structure is an example of structure that is not discoverable. A sparcity is sort of discoverable and have generic methods that do it, so okay. Matrix product, that's C = AB. Now here the number's approximately MPN. So it's the product of the sizes like this. That's what it is if it's large. Now if A or B is sparse, it can be much, much less.

By the way, this is – it's not at all obvious, actually, when you have sparse matrices. If you really want to work out good data structures for storing and manipulating sparse matrices, it's quite complicated. And a lot of the data structures you think of immediately are not that great. I mean the most obvious one a person might think of would be a key val pair, triplets, well, key val pairs, and it would look like this. That's how you might imagine is a whole bunch of things like this to be key and then val paired, you know, something like that. That's how you might imagine. They're probably the most natural for a person.

Seems like that's a very poor data structure for this, and so the ones that are actually used vary from simple, but sort of practical. That would be – you'll hear column compressed. What's the last term on that? Column compressed format, or something like that. You get column compressed format, and it's just some weird data structure that is kind of like a data structure that's a cross between simple and close enough to machines that you get something that's – but then you get all sorts of exotic data structures for storing sparse matrices, which I won't go into. But if you look at this area, you'd find out about them, okay.

Actually, this is a good time for me to point something out about this because it's a very good thing to know. Let's take matrix matrix multiply. Okay, everybody here knows how to write a code for this like in C. It's nothing, right? All you do is you take A and B, any language, doesn't matter. You take A and B, and you're gonna calculate CIJ, or let's do this. You say for I = 1 to N, for K = let's make them all end by N because I don't really care. Values 1 to N and for K = 1 to N, you're gonna calculate CIK. And so you simply walk across A, down B, and add these up.

So you get like six, eight lines of C, something like – everyone could write that right here, right. And you compile it and you will have a little function called [inaudible] A, B. Everybody got it? And it will multiply matrices, you know. It's not exactly a sophisticated computation you're carrying out, right?

Okay, so now let me ask this. I know I mentioned this is 263 if you took that class, but let me ask you. Can you imagine, there doesn't look like there's a whole lot of room for creativity in the [inaudible] thing. I mean if you were like scanning some code and wanted to know where to optimize something. You probably wouldn't take [inaudible] and the ensuing like five lines of C, two of which are comments, as something that we would optimize. I mean I'm just – does this sound reasonable to everybody?

Yeah, so here's the weird part. If you use the LA Pac version of this, it could easily be ten times faster than yours and could definitely be 100 times faster. Now, by the way, there's no structure being exploited. Total number of flops that go down in each case, identical. It's whatever it is. In a two man matrices, it's like N cubed or something, okay. Everybody, you have to let this sink in what I just said because it's ridiculous. Everybody see what I'm saying? Now I'm gonna ask you this. This is just a fun question. Why? What does LA Pac do that you didn't in your six line C program?

Student: Block.

Instructor (Stephen Boyd):It blocks, okay. So, in fact, what LA Pac did is it will actually break this up into blocks and then multiply blocks at the same time. Why does that work? Because some blocks are optimized. The size is optimized perfectly to do register register calculations, okay. And the next block is optimized for your level one cache and so on, so forth, okay.

By the way, if you don't know what I'm talking about, that's fine, but whether you know what I'm talking about or not, you must know the following, that when you get into numerical computing, there's things that just look like, you know, if you don't know about this and someone tells you that there's a good and a bad version of something that multiplies two matrices, right. It's just not obvious that it even makes any sense, but it's absolutely the case on modern computers, so that's how it works, so okay. And I'll say more about that later, but that's – this is the first thing that maybe if you don't know about this field, this is not obvious.

Okay, so let's talk about linear equations that are easy to solve. We'll start there. So well, look, if A is diagonal, it's very easy to solve AX = B because you just simply divide by the diagonal entries, so that's nothing, and that takes M flops, okay. If a matrix is lower triangular, that's the first one that's not totally obvious. That's, you know, it looks like this, A11, A21, A22, A31, A32, A33. You know, this thing times X1, X2, X3 equals B1, B2, B3, like that, okay.

How do you solve it? Well, you first go after X1 because this first equation is A11, X1 equals B1. You divide and get it. Once you know X1, you go to the second line and it says A3, A21, X1, but you just calculated X1, so you substitute it back in there, okay. So you actually calculate X1 and then you substitute it forward. This is called the forward substitution algorithm, okay. And what happens in this case is you successively calculate X1, X2, X3, X4. Once you've gotten down to here and you know everybody above here, it means that you know these guys, you multiplied by these, and that's just a number. And it goes on the other side and you divide, so it's very simple.

By the way, this thing works for block methods as well. These could be – the A's could be block lower triangular. You could also do – now, I can't use this. This code wouldn't work, but had I written this the right way, as A22 inverse here. Then my notation would have overloaded correctly in the block case, okay.

So this is – now if you do this if it's dense, this takes N squared flops with – this takes N squared flops here to work because you work out. It's basically, you know how these work. It takes like one flop. Then it takes three, then five, then seven. And you add those numbers up and you get N squared, okay, because you go, you keep adding anyway.

I'm sure you know how that works, okay. By the way, if A is sparse, then this forward substitution basically depends on the number of non-zeros in A, very, very fast, okay, if A is sparse. So forward substitution becomes fast. If a matrix is upper triangular, you just solve from the last element up and that's called backwards substitution, okay.

Well, there's lots of other. I mean you can have an orthogonal matrix. If a matrix were orthogonal, then basically the inverse is transposed, so you multiply by the transpose, and that's for general. But, in fact, often in orthogonal matrix is stored with a different data structure.

One common data structure, in fact, would be a product, just for example, of matrices of this form. That's a very common data structure used to describe an orthogonal matrix. And this – you have norm U equals 1. If you do that, then to multiply A transposed here by a vector, by the way, that's a perfect example of a matrix vector multiply.

This matrix, of course, in general, is dense, right. So if you form this outer product, it's gonna have – it's gonna spray non-zeros over the whole matrix, right. So that'll be a dense matrix and the matrix vector multiply will cost you order and squared. I think I got that right, okay. But here, if you exploit the fact that this is identity plus rank one and calculate it this way when you're past a vector, you first multiply by U. You calculate an inner product, then multiply by U, and then subtract that from your original one. It's an order N.

So matrix vector multiply for this structure of matrix is gone down from N squared to N here, okay. Another example would be a permutation matrix and it's actually – this is a great one. It's interesting. If you have a permutation matrix, so in the traditional 1960s term, you'd say there's no flops because you don't do any floating point calculation. You're just doing index tricking.

The funny part is actually nowadays this would be – this is exactly what you don't want. This is – I mean if you're actually calculate, it's like data movement and stuff like that. So this – in modern times, this would be a relatively expensive operation and it runs up according to the metric we're using zero bill because it's permutation.

So, okay, now I can tell you about the big picture of how you solves AX = B. Well, the general scheme is this. You take the matrix A and you factor it as a product of simple matrices. These can be diagonal permutations, orthogonal, upper triangular, lower triangular, block upper triangular, and these kinds of things. So you factor it this way. And then you compute A inverse B as the product of these inverses in reverse order like that, okay. You multiply it by B.

So you actually – let me just put the parenthesis in the right place. You do this, then that, and so on, up to that, okay. So you factor and then you solve from the – you solve actually from the first one backwards is what you do, something. I think I have it right, something like that. That's how you do it because each of these operations you can do fast, so that's how you do this.

Okay, so and in each case you – by the – people write things like A inverse B because that's what you write in math, but often, when people say, "Compute A inverse B," that's just sort of understood to mean no that you actually compute A inverse and multiply it by A. It means that you call a method, which is something like solve A, B. But it's just weird to write that, so people write A inverse B, and it's simply known that you wouldn't actually compute it.

You wouldn't take the pseudo code literally and actually form A inverse multiplied by B, okay. So, okay, and actually, once you know this, you already know something that is not obvious. Well, you already know one other thing that's not obvious, that basically LA Pac will beat the hell out of your six line C program or can. That's another thing that's, I believe, not obvious to multiple two matrix. Do something as stupid as multiply two matrices, LA Pac properly configured and optimized will beat you like by a wide, possibly very wide margin.

Okay, second thing that's not obvious, again, if you don't know this material, and it has huge consequences, and it's this, and it's really very cool. It's this. Suppose you want to solve a set of equations with the same matrix, but multiple right-hand sides, okay. That comes up like all over the place. Now those, for example, I don't know. AX = B really represents solving a set of circuit equations. B is a set of current sources injected into a circuit. X is the set of potentials that at the nodes in the circuit. So that's what AX = B means.

All right. In that case, it says suppose you want to – and basically AX equals – solving X = B is doing a circuit – is a circuit simulator. It basically says, "Inject these current. If I inject these currents in the circuit, please tell me what the potentials would be at the nodes." That's what solving X = B is. It's a sim.

Now once you know about factor solve methods, you know the following. If you have multiple sims, so you're saying, "You know what? I would like to know what's the potential distribution in the circuit with this current excitation and also this one and also this one and that one too, okay." The naïve way is simply to call the sim code four times, okay. But, in fact, that's silly because the factorization on the matrix only needs to be done once.

So once you have actually factored a matrix into easily solved systems, the – that is amortized across all the solves you do. And so it is just a totally weird thing. You can go and do this is mat lab, although just because it's using a LA Pac. You can do the most amazing thing. You can just go and you can time that. A has to be reasonable, otherwise it's too fast and the interpreted overhead. You know, so you could time that and then you could time this.

What's the time difference gonna be for dense matrices? I can tell you that the factor [inaudible] and the back solve you now know is N squared. What's the time difference between these two gonna be? What's it gonna do? It will actually do the right thing here. What will it do?

It will factor A once and then do two back substitutions, back solves, one with B1 and one with B2. So this is an order N cubed algorithm. Actually, or I should say quite specifically, it's N cubed plus something N squared. And this would be N cubed plus like 2N squared. What's the difference between these two? They're the same and you will, in fact, find that. So it's the most ridiculous thing.

If someone walks up to you on the street and says, "I need to solve once set of 1,000 variables, 1,000 equations." And they say, "You know what? Actually, I have ten. Same set of equations, ten right-hand sides. Please solve it for me." Guess what the time is? Identical. It's zero. It's insignificant. It's the same. So again, this is not obvious, but now you know it, and it has to do with the factor solve method. I mean this will be more clear when we get to actually how this is done, but that's it.

These things are just like not – you know, if someone tells you that it's – to calculate A inverse B1 up to A inverse BK, is that then if K is less than or less than N, like if it's order less than N. In this case, like if it's ten, then this – the cost is actually identical to calculating just one. It's weird. These are not – I mean, yeah, these are not obvious things and I think a lot of people don't know them. It just kind of doesn't make any sense.

So the cost of multiple solves is one factorization plus M solves, okay. We'll get to that. We'll see. Okay, now let's talk about the factorizations. I won't get very far, but I'll say a little bit about it. The most famous factorization actually just comes from gaucian elimination and it's this. Any non-singular matrix you can factor into a permutation, a lower triangular, and an upper triangular matrix period. And, in fact, if you want, you can even insist that the – either the L or the U has ones on the diagonal.

So this permutation – I mean this factorization is by no means unique, nor, by the way, are we gonna look at how it's done. If you take the ten week version of the thing we're gonna cover in a lecture in a quarter, you would know all the details of how you actually calculate this. Now the permutation matrix is absolutely required. So I can actually easily make up a matrix A. I shouldn't have said that because I actually can't. If I had to do it right now, I probably would fail. No, I might be able to do it, but I'm not gonna try.

You can easily make up a matrix A that's non-singular, but which cannot be written as LU. So you may have to permute, in this case rows, to make it – so the P here is not an extra thing. Actually, the P, if you take one of these classes on numerical methods, the P is chosen for two things. First of all, there's the mathematical issue that it's just not true that all non-singular matrices have an LU factorization without the P, number 1. Number 2, it's chosen so that round-off errors obtained when you calculate these things don't kill you. So that's the real reason that the P is used, but okay.

Now the cost of that factorization is 2/3 N cubed flops. Okay, so here's how you solve equations and it's roughly – actually, I mean it's not roughly. This is kind of – this is sort of what mat lab does by calling LA Pac codes, I might add. It works like this. Yeah, the X = B. A is non-singular. So it first carries out a PLU, but it's called an LU factorization. The P is understood. So if someone says it's an LU factorization, they mean PLU.

So an LU – you do an LU factorization, that's 2/3 N cubed flops. And then what you do is you – well, you start by solving P. You permute the right-hand side. Then you do forward substitution and backwards, and each of these cause N squared flops. So this is negligible, well, except if N is like six or eight. But then, in that case, these things are – different rules come into play when N is six or eight.

So, but you know if N is whatever, 100 or 50 or more, then it's just – basically it's just order N cubed. So the cost of solving, now you know the following. For dense equations, the cost of solving one set of linear equations is N cubed. As a single data point, my laptop last night, N = 1,000, half a second, okay.

And the cost of solving eight sets of linear equations with the same coefficient matrix would be on my laptop what? Yeah, it'd be basically unmeasurably – it would be you couldn't measure the difference. It would be the same. And the reason is you do one factorization and then once you've got the factorization, you do back solves ten times or something like that.

So, okay, so I think we'll quit here and then continue on Thursday. Let me remind you. The section that would have been yesterday is tomorrow.

[End of Audio]

Duration: 77 minutes

ConvexOptimizationI-Lecture14

Instructor (Stephen Boyd):Just a quick poll: I just wanna make sure – how many people have tried the image interpolation problem? And how many actually had trouble with it? One, two – okay. Unspecified trouble, I think is what they're –

Student:[Inaudible].

Student: Yeah. I think it was – had something wrong with the norm function.

Instructor (Stephen Boyd):Okay. No, there's nothing wrong with the norm function.

Student: That was an exaggerated one.

Instructor (Stephen Boyd):Oh.

Student: Like, I was trying to do –

Student: Yeah, it gets a slightly different response than if you just do the sum of squares.

Student: Right.

Instructor (Stephen Boyd):Oh, you do?

Student: Yeah.

Student: Yeah.

Instructor (Stephen Boyd):How different?

Student:[Inaudible]. [Crosstalk]

Student:[Inaudible].

Instructor (Stephen Boyd):Really?

Student: Yeah. I mean, norm looks good, but it's not –

Instructor (Stephen Boyd):Wow. Okay. You know what you should do if you don't mind?

Student: What?

Instructor (Stephen Boyd):Send your code and your build number to the staff website. I mean, the full code and the CVX build number – which I should probably mention about how you're supposed to vote early and often or whatever. You may wanna download –

redownload this CVX stuff every couple of weeks. Not that anything really has changed or anything. No, no. It's things like there's a comment in the user guide that was wrong or there's something mid – tiny, tiny things. But just every two weeks, something like that, when you – you can think of it when you clear your cash or something on your browser. Just download another CV just for fun. So I –

Student: How do I know when to do it?

Instructor (Stephen Boyd):How do you what?

Student: Know when to do it?

Instructor (Stephen Boyd):It's easy enough. You just look and see what the build number is. It's fine.

And you don't have to, right? I mean, some of these – most of these builds are just, like, fixing commas. Okay? So it's not –

Student: Could we just rerun a [inaudible]?

Instructor (Stephen Boyd):You'll figure it out. Yeah. It'll work. It'll work fine.

Okay. But actually, please, if you actually do find what you believe to be an error – and some people have found a couple – actually, mostly in the documentation – actually, e-mail us. And if you're finding inconsistencies and numbers coming out, we – actually, we wanna know about it. So [inaudible] just passively go, "Okay. Now it's working," and move on. Please let us know. Some are not – can be explained, or there's nothing anyone can do about it. Others probably need to be addressed. So – Okay. Back to numerical linear algebra whirlwind tour. So last time, we got as far as the LU factorization, which is also – I mean, it's the same as Gaussian elimination. So this is Gaussian elimination, and it basically says this, "Any non-singular matrix you can factor as a permutation times a lower times an upper triangular matrix." Actually, this, of course, is highly non-unique. There's zillions of factorizations. But for the moment, we don't need – we're not gonna even cover why you'd pick one over another or something like that. That's a topic you would go into in detail in whole classes on this topic. But the point is that once you have affected this – once you've carried out this factorization, you solve AX equals B by, essentially, you factor, and then you do back substitutions or forward and backward substitutions. And a very important point about that is that once you've factored a matrix and the cost – if the matrix is dense and you're not gonna exploit any structure – is N cubed. The back solves are N squared. And actually, what this – now you know something that's not obvious at first sight. I mean, you just have to know how this is all done to really know this because it's kinda shocking. It says, "Basically, the cost of solving two sets of linear equations with the same matrix is actually equal." More or less – I mean, for all practical purposes, it's identical to the cost of solving one. Okay? And that's not obvious, I don't think, because if someone walked up to you on the street and said, "How much effort does it take to solve two sets of linear equations compared to how much it costs to solve one set of linear equations?" I think a normal person would probably answer, "Twice." That's wrong. The answer is the same. Okay? And that's where factor-solve method – there are other methods where the number is twice. It couldn't be any worse than twice, I suppose. Right? But the point is it's the same.

Student: Could you just put the two in one system? Is that what you're saying? Like, just to stack the –

Instructor (Stephen Boyd):Nope, nope. That would make it much worse. If you stacked it like that, you'd have a 2N by 2N –

Student: Yeah.

Instructor (Stephen Boyd):And 2N cubed – you'd be off by a factor of eight. So you'd be eight times more.

Student: So what was your argument for being two – less than two?

Instructor (Stephen Boyd):That's all. One factor, two back solves. Okay. Now we need to get to something very important. A lot of you probably haven't seen it. It's probably – it's one of the most important topics, which I believe is basically not covered because it falls between the cracks. It's covered somewhere deep into some class on the horrible fine details of numerical computing or something like that, I guess. I don't think it's well enough covered, at least from the people I hang out with – not enough of them know about it. And it has to do with them exploiting sparsity in numerical algebra. So if a matrix A is sparse, you can factor it as P1LUP2. Now here, you're doing both row and column permutations, and you might ask – I mean, you don't need the P2 for the existence of the factorization. All you need, in fact, is the existence of P1. You need P1 because there are matrices that don't have an LU factorization. Though the P2 here really means that there's lots and lots of possible factorizations. The question is why would you – why would you choose one over another? And I'll mention just two reasons – again, I'm compressing six weeks of a class into 60 seconds, but one reason is this: you would choose the permutations – the row and column permutations in your factorization. First, you'd choose it so that the – when you actually do the factorization, it's less sensitive to numerical round off errors because you're doing all this calculation in floating point arithmetic. Right? Actually, by the way, if you're doing this symbolically, then you don't need to – this first issue doesn't hold. But of course, 99.99999 percent of all numerical computing is in doubles – [inaudible] floating points. You have floating points. Okay. Now, there's another reason, and this is very, very important. It turns out that when you change the permutations – row and column permutations, the sparsity of L and U is affected. And in fact, in can be dramatically affected. So if you actually get the row and column orderings correctly – that's these two permutations – the LU that you will get here – the number of non-zeros in them when they're sparse – by the way, I should add something like this: that once P1 and P2 are fixed, then you normalize L by – or U by saying, for example, the diagonal entries of either L or U are one – then this becomes unique – the factorization at that point. Otherwise, there's some stupid things you could do, like for example multiply L by two and divide U by two. But once you – then you – then they become a canonical form. They're fixed, and so once you pick these, there's only once choice for L and U once you normalize either L or U. So in fact, the sparsity pattern will be determined by P1 and P2 here. And if you – if these are chosen correctly, you will have a number of non-zeros in L and U that is small. If you choose them poorly, it'll just be enormous. And this is the key to solving, for example, 100,000-variable, 100,000-equation problem: AX equals B. Obviously, you can't remotely do that if it were dense. You can't even store such a matrix. Okay? Period. And if you could store it – well, I guess if you – you could store it and, I mean, if – theoretically, you could, but it's a big deal, and you'd have to use a whole lot of your friends machines and all sorts of other stuff to do this. I mean, it's a big, big deal to do something like that. However, if that 100,000 by 100,000 matrix is sparse – let's say, for example, it has ten million non-zeros entries, which means roughly – this is all very rough. If you have 100,000 by 100,000, you have ten million things – it means you have about 100 – on average, 100 non-zeros per row and 100 non-zeros per column – just roughly. Well, I guess that's not rough. That's exact. When you say average – so what that means is something like this: it says that each variable only enters into – on average, each of your 100,000 variables enters into, on average, 100 equations. Each equation involves, on average, 100 variables. And by the way, if you still – if you think that's an easy problem to solve, it's not. So I mean – however, that can be solved extremely fast if P1 and P2 can be found so that these factors actually have a reasonable number of non-zeros. If you start with ten million non-zeros – if the L and U – if you're lucky, L and U will have about on the order of ten million, 20 million on a bad day. Thirty million would be – you're still no problem – just not even close to a problem. If you can't find P1 and P2 that end up with a sparse cell, then it's all over. But in many, many, many cases, there's gonna be very simple – shockingly simple and naïve algorithms. We'll find permutations that give you a good sparse factorization. Yeah?

Student: How do you know, when you're finding P1 and P2, whether it can be made less sparse?

Instructor (Stephen Boyd):You don't. You don't. You don't know. So what you do is you run – it depends on what you're doing. You run at – you'll actually attempt to solve it, and then you'll be told – I mean, it depends, but actually, it'll be really simple. If it just doesn't respond, like, for the – and you can look at your memory allocation and watch it go up and up and up and up, then you'll know it's not working.

But it's easy enough to know ahead of time. There's actually two passes – I'll talk about that in a minute. The first pass is called a symbolic factorization, and the symbolic factorization, you only look at the sparsity pattern of A, and you actually attempt to find P1 and P2.

In real systems, what would happen is this: the symbolic factorization would run, and by the time it's gotten to some upper limit, like 100 million or a billion, let's say – it depends on, of course, what machine you're doing this on. But after it gets to a billion, it might quit and just say, "There's no point in moving forward because the L and U I'm finding already have 100 million entries."

Something like that. So that's how you do it in the real world. It'd be two phases for that. That's how it would really work.

The way that would work in Matlab – which, again, let me emphasize Matlab has nothing to do with the numerics. It's all done by stuff written by other people. They would just, I think, stop. I mean, you just write A/B, and it just wouldn't respond. And that would be its way of telling you that it's not finding a very good permutation.

I'll say more about this in a minute.

One thing that's gonna come up a lot for us is Cholesky Factorization, not surprising. So that's when you factor a positive-definite matrix. So if you factor a positive-definite matrix, you can write it as LL transpose – L transpose – I mean, obviously, what this means is it's an LU factorization.

Two things about it – you do not need a permutation. Every positive-definite matrix has an LU factorization. In fact, it's got an LU factorization with L equals U.

By the way, just to warn you, there's different styles for this. Other people that use – this is lower triangular. You can also do it as, I guess, let's see – upper triangular – there's an upper triangular factor. There's different ones, so I guess you just have to check. In fact, who has recently used – what's the chole in LA pack? That would be the standard. What does it [inaudible]?

Student:U.

Instructor (Stephen Boyd):In the upper triangular? Oh, okay. So sorry. We're backwards from the world standard. The world standard would be upper triangular, but everything works this way.

So this is the Cholesky Factorization. It's unique, so there's a unique – and by the way, there's no mystery to these factorizations. I'm not gonna talk about how they're done, but there's absolutely no mystery. I mean, you basically just write out the equations and equate coefficients. I mean, I don't want you to think there's some mysterious thing and you have to take this ten-week course to learn what a Cholesky Factorization – it's really quite trivial.

To write down what it is – the formula for it is extremely simple. What you should know after our discussion last time in multiplying two matrices is that little formula we write down – that's not how it's done. It's done by blocked and hierarchies and all that kind of stuff, which exploit memory and data flow and things like that.

So let me just show you what I mean here. If you write A11, A12 – you know what? That's good enough. A22 – something like that. You simply write this out that this is L11, L21, L22 times the transpose of that, which is L11, L21 and then L22 – like that. And now, this is the Cholesky Factorization, and you just wanna find out what are these numbers? Well, okay. Let's start here. Let's take the 11 entry. We notice that this times that is – so we find L1 squared is A11. Everybody following this? So obviously, L11 is square root A11. Okay? That's the first one. Now that you know what L11 is, you just do the same thing. You back substitute, and you look at the next row – would be this one times this one would be L11, L12 equals A12. And now you divide out. Do I have to do this? I don't think I have to. I mean, so this is totally elementary for you. You just write it all out. And so the existence and formulas – in fact, lots of formulas, obviously all equivalent for the Cholesky Factorization – you can derive this way. But just – and then you could write, by the way, a little chole in C, and it would be about five lines. Again, like matrix multiply, but your chole would be beaten like crazy by the real ones, which would block it up into little blocks optimized for your register sizes and cashes and things like that. Okay. All right. So that's the Cholesky Factorization. The way you solve – and here you get one third, and this is kinda predicted – the factor of two is totally irrelevant in flop counts. As we – as I mentioned last time, you can be off by a factor of 100 comparing real computation time versus flop count time. So two things with an equal number of flop counts, like for example, your amateur chole or your amateur mat molt will be beat – could be beat by a factor of 100 by one that does the right thing. So – and they have exactly equal number of flops. Actually, it could be more than 100. You could arrange it very nastily to – you arrange N to be just big enough to have a lot of, like, cash misses and things like that. And we could arrange for that number to be almost as big as we like. Okay. So when you do a Cholesky Factorization, not surprisingly, A is symmetric. A basically has about half the number of entries as a non-symmetric matrix. So the fact that the work is half is hardly surprising. But again, the factor of two is totally irrelevant. Nothing could be less relevant. Okay. So this is how you do Cholesky Factorization. And there is one thing I should say about this, which is good to know, and that is this: when you do a dense – when you do a non-symmetric LU factorization, the P is chosen primarily so that numerical round-off does not end up killing you as you carry out this factorization. It doesn't end up – you end up factoring something, but it's not – the product of the things you factor are not equal to the original matrix or something like that so your answer is totally wrong. So the P is chosen. That means that this P is often chosen, actually, at solve time. So when you – you look at – you have to look at the data in the matrix A to know whether you should do a – that's called pivoting, by the way. By the way, you – obviously, if you – you don't have to follow everything I'm saying, but it's important to know some of these things. By the way, that slows down a code. Right? Because if it's got branches and things like that – and then if it's got to actually pivot and stuff like that, you have data movement. On a modern system, this can be very expensive. Okay? So –

Student: Why did you say [inaudible] fewer elements in the positive [inaudible] case? It seems like they're the same size.

Instructor (Stephen Boyd):Yeah, they're the same size. But this is a general one. So yeah, if you get N squared entries to tell you what it is – this is symmetric. So you really only have half the increase.

Student:[Inaudible].

Instructor (Stephen Boyd):Okay. Now – so here, you would choose this – the P dynamically. So actually at – when you factor a specific matrix. Here, you don't need it, and in fact, it's actually well known that just straight Cholesky Factorization actually will – is quite fast. And actually, it's quite – it's stable numerically. In other words, it's known that this is not gonna lead to any horrible errors.

This means, for example, if this is small, like 100 by 100, you get something that's very, very fast. And also, it's got a bounded run – I mean, it's got a bounded run time. It just – it will be very fast. Right? Because there are no conditionals in it, for example. Okay.

Now we're talking about sparse Cholesky Factorization. That looks like this. Here, you of course, preserve sparsity by permuting rows and columns. Okay?

So of course, any permutation you pick – if – well, I can write it this way. P transpose AP equals LL transpose. So the other way to think of it is this: you take A, and your permute is – you carry out a permutation of its rows and columns – the same one. Then you carry on a Cholesky Factorization – by the way, it's absolutely unique – the Cholesky Factorization – provided you normalize by saying that the diagonals are positive. Otherwise, it's a stupid thing.

Question? Okay.

Okay. So this – so since you can – I can permute the As any way I like, and then – I'm sorry, compute the rows and columns of A and then carry out a Cholesky Factorization, when A is sparse, this choice of P is absolutely critical. So if you give me a sparse matrix, and I pick one P, there's just a Cholesky Factorization. Period. And that Cholesky Factorization will have a number of non-zeros in L. In fact, if you choose P randomly, L will basically be dense for all practical purposes. Okay?

And then it's all over. Especially if your matrix is 10,000 by 10,000, it's pretty much over at that point.

Yeah?

Student: Could you [inaudible] solve the equation using Cholesky? You need to know the matrix is positive-definite. Right?

Instructor (Stephen Boyd):That's correct. Right. So –

Student:[Inaudible]. [Crosstalk]

Instructor (Stephen Boyd):I can tell you exactly how that works. So in some cases, it's positive-definite because, for example, it's a set of normal equations. That's very common. Or it's gonna be something we'll talk about later today. Maybe we'll get to it – Newton equations. Right.

So a lot of times, it's known. Now in fact, the question is what happened? Obviously, the matrix has a Cholesky Factorization. It's positive-definite. In fact, if L is – especially if this is the case, it's positive-definite. What happens if it's not? That's a good question. And here – I didn't get very far in my Cholesky Factorization, but in fact, what would happen is this: at each step of a Cholesky Factorization, there's a square root. Right? You really did the first one. The next one would be square root of – well, that's basically the determinant. Okay? It'd be the square root of the determinant would be the next one.

What would happen is this: that square root – if all those square roots – if the arguments passed to them are all positive, then you will have a Cholesky Factorization. If that matrix is not positive-definite, one of these square roots – you're gonna get a square root of a negative number. Okay?

And then several methods – there's actually different Cholesky Factorizations modified. In fact, some – I forget the name of it, but there's a type of Cholesky Factorization which you might want, and it works like this: right before it takes a square root, it has a test that says if this thing is negative, it throws an exception.

By the way, it can even be arranged to throw – to actually calculate a vector Z for which Z transpose AZ is negative, which of course is a certificate proving A is not positive – or less than or equal to zero. Whatever. So it would do that.

That's one way. Unfortunately, a lot of the other ones – because these are built for speed – it'll just throw some horrible exception from the square root or something like that.

So that's – there's a modified – in fact, Cholesky is, in fact, the best way to check if a matrix is positive-definite. You find that number. You don't calculate the IG or anything like that. You just run a Cholesky. You don't want a basic Cholesky, which will dump core, and then that will be your – then you have a system called – anyway. No. I'm joking.

But the way it would work is this: you'd run with this modified Cholesky that, when it encounters a square root of a negative number, does the graceful thing, which is to return telling you, "Cholesky didn't succeed," which is basically saying, "Not positive-definite."

So that's the best way to check.

Okay. Let's go back to this one. This permutation. Now, in – so here's the way this actually works, and I'll explain how it works. The permutation here is generally not chosen in Cholesky Factorization. What I'm about to say is not quite true, but it's very close and is a zeroth order statement – it's good enough. You don't choose the permutation in a sparks Cholesky Factorization

to enhance numerical properties of it.

Now technically, that's a lie, but the first order – it's pretty close to the truth. It's a perfectly – it's a zeroth order. It's a good statement.

You choose the P to get a sparse L. That's what you do. Okay? And what happens then is this: there's a whole – I mean, this actually – there's simple methods developed in the '70s where you would choose these Ps – and I mean, they have names like approximate minimum degree. Then they get very exotic. Then there'd be ones like Reverse Cuthill-Mckee ordering. Then you get to very exotic orderings.

So, that's all fine. That's all fine, but now I'll tell you as most of you would be consumers of sparse matrix methods – in fact, you don't know it, but you have already been very extensive consumers of sparse matrix methods. Okay? Because when you take your little baby problem with 300 variables and CVX expands it – I don't know if you've ever looked, but if you looked, you'd find some innocent little problem you're solving with ten assets or something like that – if you look at it, it'll all of a sudden say, "Solving this problem with 4400 variables."

Well, trust me. Oh, systems of – when it says 27 iterations, I promise you equations of the size 4400 by 4400 were solved each iteration. Okay? You would have waited a whole long time if this wasn't being done by sparse matrix methods. So you're already consumers – big time consumers of sparse matrix methods. So what I'll – what I wanna say about this is that even very simple – by the say, all the things that you did, although it's in the solver, which I don't know the details of – those were – the orderings were very simple ones. They were things like approximate minimum degree. I mean, just really simple, and for most sparsity patterns, this things obviously work fine. So that's what this is. Now the exact value really depends in a complicated way on the size of the problem, the number of zeros and the sparsity pattern. So actually, it's interesting to – I wanna distinguish the dense versus the sparse, so let's actually talk about that. So – and I'll just talk on a laptop little PC – something like that. Dense is, like, no problem. You can go up to, like, 3,000. I mean, just no problem. [Inaudible] scale something like N cubed. You already know that. And it'll just work. It'll start allocating more memory and start taking, like, a macroscopic time, but yeah. It'll just – it's gonna work. There's essentially no variance in the compute time. It's extremely reliable. It's gonna work. It's gonna fail if the matrix you pass in is singular or nearly singular, but then you shouldn't have been solving that equation anyway. So you can hardly blame the algorithm for it. So these things – for all practical purposes, the time taken to solve a 3,000 by 3,000 equation with 3,000 variables – the time taken is just deterministic. By the way, at the microscopic level, that's a big lie. But it's good enough. And the reason it's a big lie is very simple. Actually, depending on that matrix – the data in that matrix, different orderings and pivoting is gonna do – and if you have no pivoting, it might take less time or whatever. And you have a lot of pivoting and all that, it might take more. But it's absolutely second order – just not there. Okay? So that's the deal with – so there's no stress in solving a 3,000 by 3,000 system. Zero. Absolutely zero. It just works. You can say exactly how long it's gonna take ahead of time, and that's how long it's gonna take. Period. Now you go to sparse. Now – well, it depends. There's actually two passes. It depends on the sparsity pattern. So let's go to a typical example: 50,000 by 50,000 matrix. So you wanna solve AX equals B, 50,000 by 50,000. Now some people, you can make a small offering to the god of sparse matrix – orderings that some people say is effective. But the point is, there's no reason to believe that you can always solve 50,000 by 50,000 equations. Okay? And in fact, there's ones that you can't. And it just means that whatever sparse solver you're – whatever ordering method is being used, typically on your behalf – I mean, unless you do go into large problems. If you do large problems, you have to know all of this material – every detail. Okay? If you don't work on huge problems, then this is stuff you just have to vaguely be aware of on the periphery. Okay. So in this case, 50,000 by 50,000 – there's a little bit of stress. You type A/B, it might come back like that. You'll know examples where it'll come back like that soon. Okay? Or it might just grind to a halt, and you can go and watch the memory being allocated and allocated and allocated, and it just – at some point, you'll put it out of it's misery. So that's what – however, the interesting thing is this. Once that's done – if you do it in two steps – if you do a symbolic factorization first, you're gonna know. You'll know. Then after that, it's – for matrices of that sparsity pattern, it's absolutely fixed. Right? So this value that has a lot of implications in applications. If you write general purpose software that has to solve AX equals B where A is sparse, you don't know. Basically, you deal with it every time it's fresh. If I pass you an A matrix, the first thing you do is you do a symbolic factorization. You start going over it, and first, you look at it and you go, "Well, it's sparse. All right. Let's see what we can do with ordering." And then you do the ordering. You do the symbolic factorization. You know how many non-zeros there are gonna be. At that point, if you wanted to, you could announce how long it's gonna take to solve that problem because once you know the sparsity pattern of L, it's all over. You know everything. In fact, just the number of non-zeros is a good – question?

Student:[Inaudible] symbolic factorization?

Instructor (Stephen Boyd):Well first of all, that's in the appendix, which you should read. So let's start with that.

Student: Appendix B?

Instructor (Stephen Boyd):Well, yes. Okay. Some people are saying it's appendix B. It's one of the appendices. There's not too many, so a linear search through the appendices wouldn't – it's feasible, I think. So the appendices where this is covered in more detail, but you have to understand even that is like condensing a two-quarter sequence into 15 pages or something.

So – okay. So the interesting thing is if you're gonna solve a set of equations lots of times – like, suppose you wanna do medical imaging or you wanna do control or estimation or some machine learning problem or something like that, but you have to keep solving the same problem with the same sparsity structure. At that point, actually, you could invest a giant pile of time into getting a good ordering. So it could take you – you could run this whole communities that do nothing but calculate ordering. You could actually calculate your ordering for three days, and when you get – by the way, if you get an ordering that works and has sparse Cholesky Factorization, it's no one's business how you got it. Right? Like a lot of things, it's just – it's fine. So you could do that. And then, it can be fast. So – okay. All right. Well, that's an – oh, let me just summarize this. For dense matrices, totally predictable how much time it is. Sparse – depends on the sparsity pattern, depends on the method you or whatever is solving this equation on your behalf is using to do the ordering. By the way, if it fails on Matlab or something like that because it's using the simple approximate minimum degree, symmetric minimum degree, blah, blah, blah ordering, it could easily be that if you were to use a more sophisticated ordering, it would actually solve it. Just no problem. Right? So – okay. On the other hand, it is too bad that the sparsity pattern plays a role in these methods. On the other hand, hey. You wanna solve 100,000 by 100,000 equations? You can't be too picky. You can't say, "Wow. That's really irritating. Sometimes I can do it, and sometimes I can't." So anyway. All right. Enough on that. And finally, one more factorization we'll look at is the LDL transpose. So this is just for a general symmetric matrix. Here, the factorization is this. There's a permutation. There's an L and an L transpose. In fact, they can have ones on the diagonal. In fact, you can't – what you do here is D is actually a block – it's block diagonal, and the blocks are either one by one or two by two. The essential matrix is easily inverted, by the way. If I give you a block matrix where the blocks are one by one or two by two, can you solve DZ equals Y? Well, of course you can. Right? Because you either are dividing or solving a two by two system. So – okay. And it's the – it's sort of the same story. So I think I won't go into that. Now, we'll look at something that's actually – everything we've looked at so far is just so you can get a very crude understanding of what happens. There's nothing for you to do so far. Right? So in fact, most of this stuff is done – in most systems, this is done for you. You solve a – you call a sparse solver. You can do that in Matlab just by making sure A is sparse and typing A/B, just for example. Or the real thing would be to directly call something, like, from UMF back or spools or there's all sorts of open sores packages for doing this. And you're probably using them whether you know it or not. So you're certainly using it in Matlab. If you use R, you're using it. If – anyway. So – actually, I don't know. Does anyone know? Do you know what the sparse solver is in Packagen R?

Student:I don't have one.

Instructor (Stephen Boyd):No, come on.

Student: It's a separate package.

Instructor (Stephen Boyd):Than – okay. That's fine. But it has one. Come on.

Student: Yeah, but it's not like private [inaudible] language.

Instructor (Stephen Boyd):Well, that's okay. That's fine. That's – I mean, come on. It's an open sores thing. It doesn't matter then, right? So that's fine. Okay.

Now, we're gonna talk about something that is gonna involve your action. Everything so far has been – it's just me explaining what is done on your behalf by some of these systems. Now, we're gonna talk about something you actually need to know because you're gonna need to do it. So – and these are things, by the way, that will not be done for you. They cannot be done for you. You'll have to do them yourself. So this – now is the time – up till now, it's just sort of, "Here's how things work." Now you better listen because now there's gonna be stuff you have to understand and you'll actually have to do.

So we'll just start with this blocking out a set of equations. Really dumb – I mean, we just divide it up like this. You can divide it up any way you like, and we're just gonna do elimination. So it's gonna work like this: we'll take the first row – block row. It says A11 X1 plus A12 X2 equals B1. So I just – I solve that equation by multiplying on the left by A11 inverse, which I'm assuming is invertible. Obviously, a leading sub block of a non-singular matrix does not have to be non-singular. Right? That's obvious, as in 0111 is obviously non-singular, and its leading one by one block – its A11 is certainly not.

But assuming it is, you can eliminate X1, and you just get this. That's X1. You back substitute X1 into the second equation, and in the second equation, you get A21 X1. That's A21 A11 times this junk over here. And then that plus A22 X2 equals B2, and that equation looks like this.

Now, this guy you have seen before, although you don't remember it probably. Or maybe you do. That's the sure compliment. The sure compliment just comes up all the time, so you might as well get used to it because it comes up all the time.

By the way, if you have an operator called the sure compliment, that's what – the Cholesky Factorization is, like, two lines because you just keep calculating the sure compliment of kind of what's below you or something like that.

So that's the sure compliment times X2 is equal to this right hand side.

Now – so this suggests a method for solving this set of equations, and you do it this way. You first form A11 inverse A12 and A11 inverse B1 – and by the way, I have to say something about this because this is sort of already high-level language. When you say form this, how do – by the way, how do you do it? Do you actually calculate the inverse of A11 and – it depends on what A11 is. So what this really means is the following: you factor A11 once, and you back solve for B1 and then each column of A12. Okay?

So it's understood when you see this in pseudo-code that you know what you're doing and that you don't form this inverse and all that kinda stuff. It's just understood. That's what it means.

Student:[Inaudible] in Matlab because we can wait to do that?

Instructor (Stephen Boyd):Yeah.

Student: Can Matlab store that [inaudible]?

Instructor (Stephen Boyd):Yeah. So let me – this is coming up on your homework, which we'll put on the web server. That is if it's up. Sorry about the AFS outage yesterday. I don't know how many people noticed it. Anyone notice it yesterday? I think it raises our blood pressure a lot more than yours. So I gotta learn, actually, just to just kinda – it's like a power outage and just deal with it. But it makes you very nervous. That's my, like, nightmare – number six worst nightmare is final exam – AFS outage. Or something like that. I don't know.

Okay. What were we talking about? Fact solving. Yeah, yeah. Matlab. All right. Yes. Of course.

Okay. Let's suppose that you needed to – let's suppose we had to do a Cholesky Factorization just – okay. So the way you do it is this: So you're gonna factor a matrix A – okay. Here's what you do – and this will work. So first, I have to tell you a little bit about what a backslash does. Okay? Which you could figure out anyway.

So it does a lot of things. I'll just give you what's essential for what we're talking about now. Here's what this does. First of all, it checks if A is sparse or something. But even if A is full, it looks to see if A is upper or lower triangular. Okay? If A is upper lower triangular – actually, more than that. If it can be permuted to upper lower triangular – but let's say it's just – if it's upper lower triangular, it will actually be back solved. Okay?

So for example, that means if I write A is 3,000 by 3,000, that's a big enough matrix that you will see a macroscopic difference. And I type A/B if A is full. It's gonna take order N cubed. I don't know how long it'll take – 20 seconds, 15, 12 – I don't know. Something like that. Okay?

I mean, obviously, it depends on the machine you're on. But it's gonna take a macroscopic amount of time. You'll hit carriage return, and you will wait a little bit before you get the answer. Okay?

If A is 3,000 by 3,000 and lower triangular, what will happen when I type A/B?

Student:N squared?

Instructor (Stephen Boyd):Yes. It's N squared, and it'll be – I won't even see it. I'll just – I'll hit carriage return. It'll take more time to actually go through all the – to interpret it and to instruct X or whatever is drawing your window to type into that terminal. That's gonna take way more time than solving it.

So it'll go from, like, ten seconds to nothing. We can – in fact, we can guess how much faster it'll be. It's gonna be on the order of 3,000 times faster – that's by the order. But in fact, the numbers are different by a factor of two. So it'll be about 1,000 times faster. This is my guess.

Okay. So this is the first thing you need to know. Okay.

Now, suppose you wanna do – suppose you wanna solve AXI equals BI in Matlab where you have a bunch of these. Okay? So this would be intensely stupid. I mean, it should work, but anyway – that doesn't matter. If you did this – I'm just writing something that's in between – I don't even know what it is.

But anyway, if you did this here, this thing – what do you think it'll do? It'll just – it'll look at A fresh each time and go, "Great. Let's factor that guy." Right?

It'll even recalculate the permutations fresh every time. Okay? So this is not the way to do it. Okay?

If you do this, though, it will actually do the right thing. If you do that, it'll do it. I mean, this is nothing but a batch – that's a batch solve. That's all it is because the columns of A inverse B, where B is this thing, is nothing but A inverse B1 inverse B2 and so on. So you can call this a batch solve or whatever you want.

So if you write A/B, it'll do the right thing. It'll factorize A once. It'll go through the columns and back solve.

Actually, to tell you the truth, even that is not – that's actually not what it does. But let's pretend that's what it does. It doesn't do that, but it again blocks it out and calls an LA pack code. But these are fine details.

This'll do the right thing here. Okay?

So for example, what is the run time of this versus this? What's the run time of those two?

Student:[Inaudible].

Instructor (Stephen Boyd):Identical. Basically identical. It's not identical, but it's – basically, it's the same for all practical purposes. Okay? So that's an important thing to know.

Now, we can put all these together. And now suppose you need to factor A once, but then you don't have all the things you wanna back solve at the same time. Then you can't do this trick. It's not a trick. I mean – okay. Here's what you do. Now we'll get through it, and I'll show you how to do it.

You would do something like this. I'm just gonna do this approximately. By the way, I think this will give you the upper triangular version of L, but it hardly matters.

So you do this once. Okay? And the cost on this is N cubed. Is that the comment? I think it is. Okay, there. Okay? That's an N cubed thing. Okay?

So for 3,000 by 3,000, you're gonna – there'll be a microscopically visible delay when you do this. Okay?

If by then, do you – well, let's see how I can do this. You have LL transpose equals A. So A inverse B is L minus TL inverse B. Is that right? Yeah. So here's my – ready for my code now? If I do this in store L, anywhere else from now on – if I wanna solve AX equals B, I can do it super fast like this. Let's see if I can get this right.

There. There you go. I can execute this anywhere I like with different Bs, and it will execute super fast N squared. Do you know – do you see why? The reason is – I mean, there's a lot of interpretive overhead, so the whole thing – I mean, this is a bunch of stupid overhead. Basically, it looks at this – I mean, I'm just – this is just – this is the formula for that. Right?

So it looks at this, and it says, "Okay. Let's look at L." And it looks at L, and it pokes around for a while. And then after a while, it says, "Say. L is lower triangular. Oh. Okay. I'll call the back solve instead of doing blah, blah – instead of doing – factoring it. It's already factored," or something like that.

So we'll do this. Then this will be passed to this thing, and it'll look at that, and it'll analyze the matrix as if it's never seen it before and go, "Hey. Wait a minute. That thing is upper triangular. Oh. Okay. I'll just go ahead and –."

By the way, all of this thinking and – takes way longer than to actually do it, unless the matrix is a couple thousand by a couple thousand. Right?

So – and then it's fine. But the point is that this will now execute very fast. So I think I just answered your question.

Okay. All right. So let's – back to block elimination, which is what this is called. So what you do is you form A11 – and actually, the comment I just made is very important because it's about to come up right now.

So these two, you would solve by factorizing A11 once and then back solving on A12 concatenated with B1. So this line would translate into a very compact thing. Okay.

Then you form the sure compliment. Now, to form the sure compliment, the point is that you've already calculated these columns. So you would – this is just a matrix multiply here and a subtraction. So you form the sure compliment, and you form B tilde – that's this right-hand side guy here. This guy. No more of this inverse is appearing here, but you've already calculated the sub expressions, so this is fine.

Then you determine X2 by solving this equation. That's the sure compliment

times X2 B tilde, and then you get X1. Once you get X2, you go back and you replug X1 from wherever our formula was – here it is. Right here. From this guy. Okay?

Now, the interesting thing here is you have to do the following: you have to solve A11 X1 equals this. But guess what? A11 you already factored on step one, so this last – step four is actually a back solve. It is not a factorization – the back solve. Right?

So these are very important things.

Student:I have a question.

Instructor (Stephen Boyd):Yeah.

Student: Why does the sure compliment arise? Like, what's the common thread?

Instructor (Stephen Boyd):Oh.

Student: Is it just coincidence? Or is there, like [inaudible]. [Crosstalk]

Instructor (Stephen Boyd):No, it's not coincidence at all. Yeah, there's a physical reason for it. It has lots of – I think you did a homework problem on it, anyway. I think it's – it comes up when you solve one set of equations – if you have a block set of equations, and you solve one in terms of the other. That's basically where it comes up. So – and it comes up in a lot of physical things, too, like in networks and – it just comes up all the time in all sorts of things. PDEs – it just comes up everywhere, basically. So whenever you have some linear system, and you impose some boundary conditions on some of the variables, a sure compliment's gonna come along.

It comes up in statistics, too. I mean, this is – this happens to be the covariant – this is the error covariance of estimating X1 given X2 or the other way around. Do you remember which it is? One or the other. But I have a Galician joint variable, and I estimate X1 condition – if I look at the random variables expected – the condition expectation of X1 given X2. I look at the conditional covariance – it's this thing. So just – it just comes up everywhere.

So – okay. So if we work out this block elimination and see what happens, you have a big drum roll, and you work it out, and you think it's all very clever. When you total it up, you get something like this. And we're gonna do it in terms of factorizations and solves – back solves. You get something that looks like that. Okay? And we're going to assume, by the way, that the smaller system – the sure compliment one – we're gonna do by dense methods. So to pay for the dense method, it's gonna be N cubed.

So you get something that looks like this – N2 cubed. You get this. Now, if you just plug in the general matrix A, you – it comes out, actually, not down to the leading order, but of course, as it has to be down to the flop. It's identical to what we had before. So there's no savings. There's absolutely no reason to do this. Well, sorry – in terms of flop count for a dense matrix.

Now, the truth is this is exactly how it's done when you do blocking and stuff like that. This is how you get fast because you arrange for, like, the sure compliment block size to be just right for your registers and your L1 cash and all that kinda stuff. And you get that size just right, and this thing will give you a big speed up. But that's a secondary question.

Now, this is useful – when this is – block elimination is useful exactly when A11 has extra structure that can be exploited. Okay? Now, that's – when it does, this method is gonna speed you up like crazy. So let's look at some of those.

Here's an example. We'll do a couple of examples, actually, because they're very, very important.

If A11 is diagonal, okay? That means that your matrix looks like this. I'm just gonna draw a pattern like that. Okay? If it looks like that, then you're gonna do unbelievably well, and here's why. Let's just put some numbers on here. Let's put 10,000 here, and let's put 1,000 here. Okay? Something like that.

You're gonna do really, really well. So I have 11,000 equations, 11,000 variables. But it's blocked in such a way that A11 is diagonal. Okay?

And by the way, this has meaning – this will have meaning in any application which this arrives. It basically common variables, and everybody depends on them. Every equation, everybody – if you go – if you look at every variable, they all depend on those 1,000 common variables.

But then there's 10,000 variables that don't depend on each other. And all I'm doing is I'm providing the English description of this sparsity pattern. That's all I'm doing. Okay?

So if you see this – actually, from this class onward, if you see this, some big green lights should light up. This is what you want to see. If you see something like this, you will know this will scale to unimaginably large systems. Okay? And it'll scale very gracefully.

Now, the reason here is this. You just go back and look at what we're gonna save, and you can actually even determine – I lost my – oh, here it is right here. All right. So let's actually see what happens in this case. A11 is diagonal. Well, this is, like, super cheap right here. That's, again, a non-issue. Don't even think about it.

This – by the way, here's the joke. Forming S is now the dominating thing. So in fact, if you were to – the joke, as in this problem – if you were to actually do a – if you're gonna profile the code that solves this, you would find out that, most of the time, it would be done doing matrix multiplication. It would actually be forming this. Okay? That's gonna dominate completely, and in fact, it's gonna be something like – it'll be 1,000 by 10K by a 10K by 1,000. Matrix multiply – that's it.

In the end, you'll have to solve 1,000 by 1,000 system, but that's gonna be 1,000 squared, and this one was 1,000 times – 1,000 squared by 10,000 is ten times more. So we know exactly the number of flops it's gonna take. It's gonna be 1,000 – it's gonna be multiplying this matrix by one like that. That's it. And it will be seriously fast.

If you were to take this and pass in a matrix like this to – and just write this back slash something, it's not gonna work. I mean, basically, it'll just sit there and – actually, it'll very happily start factoring this – very happily. This will not be recognized – this thing. This structure will not be recognized, and it'll start factoring things and things like that. And most of the time, of course, it's multiplying zero by zero and adding it to zero and then storing it very carefully back and then accessing it again later – pulling it back in. But it's very happy to do so. No problem. It'll just do it, and it just will never finish. Okay?

So – by the way, this is called arrow structure. So it's good. It's a good thing. So from now on, arrow – by the way, this – let's do another example. Suppose, in fact, this 10K was ten 1K by 1K blocks. Okay? I can't fit ten in there, but you get the idea. Okay?

By the way, there's a story behind this. Basically, it says, "I have 11 groups of 1,000 variables each." Okay? One of those groups connects to everybody – that's this last group. The other ten only connect to that last guy. So – in fact, you should even visualize the following graph. Right? It looks like that except there's ten of these. Okay?

So each note here on the graph is a group of 1,000 variables. This group does not depend on that one, but one of them depends on all of them. How do you solve this? By block – I mean, you do block elimination, this'll be way, way fast because how do you do A11 inverse?

A11 is 10,000 by 10,000. What's the effort to compute A11 if it's ten 1,000 by 1,000 blocks? How do you actually – well, you don't calculate A11 numbers. Sorry. How do you calculate A11 inverse A12? How do you form this expression? How fast can you – what's the cost to factorize a block diagonal matrix?

Yeah, you factorize each block. So this is ten times 1,000 cubed divided by three. That beats like crazy 10,000 cubed over three. I mean, by a long shot. Okay? So that's very important.

Okay.

Student: So are – is the moral of the story don't use A/ and just go through this algorithm in your own script?

Instructor (Stephen Boyd):No. Now I'll tell you a little bit about this. Okay. So having said all that for this example, here's what would happen. So – okay. So let me now, let's talk practically about what would happen here. It would probably work. If you had a matrix whose sparsity pattern looked like this, okay? If you did this – now, let me just explain a few things. If you just did this /B, too bad. It'll never work. And it's 11,000 by 11,000 – never, never. Okay? However, if you gave, say, Matlab or some other system the hint that this was sparse – so if you say it's sparse, all you're saying is that there's a bunch of zeros. There are a bunch of zeros. You see this giant triangle here – giant triangle here. These are all zeros. Okay? That's a lot of zeros. It's a lot of non-zeros, though, over here. Right? But if you do – then, actually, it is overwhelmingly likely that it won't do a bad job because the method will actually get all of these guys first, and then – in the ordering, you will get something – it might not be – it might not get it exactly right, but it's likely to do pretty well because this – so in fact, this arrow structure is one that a normal sparsity handling system is probably gonna recognize. Okay? So in this case, declared as sparse – do backslash – you can do your own experiments, but they're not really very interesting experiments. They're just experiments on what some people implemented in messages. I mean, it's not that interesting. Right? I mean, the important part is if you do real problems like this, you're calling these things directly. Then, of course, it's your job to do this. But now, let me mention some others where it's not obvious. Okay? Let's do one. Here's one. Let's suppose you do medical imaging, for example, and suppose you do MRI or something like that. And you have to solve an equation that looks like this. Okay? Let's make this like one million variables, and let's make this 1,000 here. By the way, I have no idea what this is, but let's just imagine that such a thing is there. And this one million by one million here – if this is, like, diagonal, you know what to do. We just did it. I'm gonna make this, though, a DFT. Okay? So it's a discrete Fourier transform. That's a dense matrix – completely dense. So you can form – well, you could attempt to form your million plus 1,000 by a million plus 1,000 matrix and actually attempt to solve it. It's completely dense. Everyone agree? Totally dense. Okay. Those sparsity is gonna help you out – the gods of sparsity – you are on your own on this one. Okay. However, how fast can you solve this system? You can't even store this, right? All you have to store is this part, right? This is pushing it, but that's your fault for going into medical imaging. You do medical imaging, you're not gonna do this on something with a couple of gigs of RAM, right? Obviously, we can work out what that is. But this is perfectly storable otherwise. You don't store this block, obviously. How do you solve this?

Student:[Inaudible].

Instructor (Stephen Boyd):Yeah. Of course. So you look at this, and you go back, and you say, "Well, what do I have to do?"

And you go, "Well, I have to calculate A11 inverse a couple of times."

By the way, you do not solve a discrete Fourier transform equation by – you can't even form – if it's a million by a million, you don't even form a factorization. It doesn't matter. You actually calculate by the inverse DFD. It's analogue N. So it's 20 times 20 times a million flops. It's way, way fast. Okay? Everybody got this?

So – by the way, these are not obvious things. They're known by lots of people, but probably more than an order of magnitude, more people who should know these things don't know them. So that's why I'm telling you these things.

So – okay. So this is very important to know. So – in fact, I cringe whenever I hear people very unsophisticated – in very unsophisticated ways talking about scaling. And it's like, "Well, I tried that. It's way too slow."

Well, anyway. By the way, the difference between this and just typing A/B – the technical name for that is – this time, they actually – that's called stupid or naïve linear algebra, and this is called smart linear algebra.

Smart linear algebra means you kinda know what you're doing, and you're exploiting structure at this level. That's what it means. So –

Student: How do you [inaudible] in English?

Instructor (Stephen Boyd):How do you communicate it – well, you – well, it might – it'll just end up being sparse. I mean, do you mean in Matlab, how do you do it?

There's a very bad way, and then there's a good way. You make it sparse in the beginning, and then it'll be preserved as you do the right operations on it.

You have to be very careful in Matlab because, like, one false move – one little character here, and boom. You have a dense matrix. You add identity to it. Now, adding identity to a sparse matrix, you got a sparse matrix. Right? Not in Matlab, you don't. So you have to know, "Oh, you shouldn't add IEYE, you have to add spy or something." I don't know. SPEYE – that would be the sparse identity.

Student:[Inaudible].

Instructor (Stephen Boyd):Right. Now, of course, that makes you wonder under what circumstances the identity not sparse, and I don't know of any. Maybe two by two case because then you could say, "Well, only 50 percent of the entries are non-zero."

I don't know. It's not my – but anyway. Okay.

So I think that's it. So basically, you wanna look for things – block things. If you look at a block, and you see a block that's easily solved, exploit it.

By the way, if what's easy to solve about it is it has to do with it's sparsity, you can just be lazy and let your sparse – let whatever – do your offering to the gods of sparse matrix factorization orderings – heuristics for the orderings, and hope that it works. But if it's dense or something like that, you're gonna have to do the work yourself. That comes up in a bunch of cases, one of which we're about to look at right now. But that's the idea.

Let's quickly talk for fun about some fast – what systems can you solve fast? I mean, we've already looked at some obvious ones: diagonal, lower triangular – we just talked about one a few minutes ago: block diagonal. How about, like, block lower triangular? Can you do that fast?

Yeah, of course you can because if it's block lower triangular, you can do back substitution at the highest level or whatever, and you'll have matrix inverses to do on the blocks. Okay?

We talked about another one, [inaudible] FD. Sorry – that's the algorithm. DFD – so discrete Fourier transform is the operation. [Inaudible] FD is am algorithm, which does it fast.

Others would be things like DCT – so Discrete Cosine Transform that you'd use in imaging. Anybody know others that are fast?

Student: Toplets?

Instructor (Stephen Boyd):Which one?

Student: Toplets.

Instructor (Stephen Boyd):Toplets, yeah. Toplets is a good one. So you have fast algorithms for toplets matrices, and they all have names. And usually, people are tortured by having to learn them every operation by operation like Levinson-Durbin is one of them, and – did they teach this in statistics? Probably. No. Okay, good. Fast – oh wait. It doesn't matter. For toplets? No. Okay.

Hunko matrices is another one, so it goes like this.

Student: Circulant.

Instructor (Stephen Boyd):Circulant – yeah, circulant. How fast can you solve a circulant?

Student:N log N.

Instructor (Stephen Boyd):N log N. Right because you take an FFT, multiply FFT back. So that – and circulant, of course, is the matrix is circular convolution. Convolution – well, that's toplets. We've already covered that.

There's a bunch of others, and it's fun to actually poke around and learn about others, like there's one called the fast Gauss transform that comes up in machine learning. That's, like, way fast.

So it's – there's a bunch of them that are, like, order N. There's some other ones that come – they're whole fields that can be described as fast methods of solving AX equals B.

For example, if you solve an elliptic PDE, you are solving a very specific AX equals B. Okay? And those can be done, like, some of them are light, and they can solve it insanely fast. It's not easy to do. It is not trivial. But for example, if you go to someone and say, "Oh, I really – I have this grid. Like, I have an image," let's say. Right? For example, suppose you have an image, and all the constraints only connect a pixel and its immediate neighbors. And you say, "I have to solve a set of linear equations that looks like that."

It'll have a very characteristic look if you look at it. It'll be, like, stripes and things like that. And if you do PDEs and things, you'll just immediately say, "Ah. Okay."

But in fact, solving that equation, that's the one-one block in an image. Let's do the image you're doing or will do or have done in homework eight – seven. What are you doing now, anyway?

Student: Six.

Student: Six.

Instructor (Stephen Boyd):Oh. Well, that shows you what we're thinking about. All right. So suppose we scale it up to a million by – 1K by 1K, so you have a million variables. You – actually, each step in that algorithm – the 20 steps you will solve – I promise you a million by million equation. Definitely.

It's super duper sparse. Right? Because each row and column will only have, like, two and three entries because you're connected to yourself. You're connected – four entries. Whatever. You're connected to your neighbor above, below, left, right. Right?

So super sparse. By the way, sparse orderings won't do so well. It does okay for your homework – or it should, anyway. It does. But it won't scale to bigger numbers very gracefully. However, if you look at it carefully, you'll realize that, in fact, solving that set of equations – that big A11 block for that problem, you know what it is? Solving an elliptic PDE on a uniform grid.

Then, you go to your friends. It's like a plus on equation or something like that. So you go to your friends who know about PDEs, and you say, "How do you solve that?"

And they fall down laughing. And they're like, "Are you kidding? 1K by 1K? We precondition. Blah, blah, blah – Fourier transform – multi-level."

I mean, whatever they do – there's probably some people here who do this. Okay? It's not easy. It's a whole field. Right? But the point is, they know how to do it. And then you – after they calm down, you say, "Great. Could you just give me the code that solves that?"

And then wherever you have to do A11 in the back substitution, you put that there. Now you're done.

So, okay. Okay. That's enough on that.

Let's look at a consequence of this. It's the idea of a structured matrix plus a low rank. This is actually very important. This is also extremely good to know. And this is also – this is something that you have to do. In other words, you better recognize this because no automatic system can do this. Period. So you have to do it.

So here it is. Suppose you need to solve A plus BC times X equals B. Normally, B and C are small. So you're supposed to think of B – this is low rank. This is, like, rank P. B is a skinny matrix. C is fat, and A is something that's quickly invertible. A could be the identity or diagonal – block diagonal, lower triangular. You take your pick. It could be solving a plus on equation. It doesn't matter. It's something. You can do fast circulant – who cares?

Okay. And you wanna solve this equation. Now, here's the problem. Generally speaking, whatever structure was good about A is almost certainly destroyed by adding – in fact, just add a dyad. Just add a little B, a little C to it, and it'll destroy anything. It'll – and of course, it'll destroy a DFD matrix, a DCT matrix, toplet structure is totally blown out of the way. Diagonal – forget it. You make an outer product, it'll spray it non-zeros everywhere. A block diagonal ruined – so that's kinda the idea.

Okay. By the way, once someone calculates this and hands you this matrix – all over. You can't – if they don't tell you it's A plus – once they multiply this out and hand it to you, it's all over. You'll never recover. You'll never get any speed out of it or anything. You're doing – you're stuck at N by N now.

Okay. So let's kinda do this. What you do is you do what we did in duality, which is you uneliminate. You introduce new variables. It's strange. So here's what you do. You uneliminate. You write this as AX plus B times CX, and you introduce a new variable called Y, which is CX. You uneliminate.

Then you write this as AX plus BY equals B, and then CX minus Y equals zero. That's that this is Y equals CX.

So, if you can – these two are completely equivalent. If you solve one, you can solve the other. Okay?

Now when you look at this, you should have the following urge: you should look at a matrix like that, and your eye – don't ignore this. Your eye should gravitate towards the minus I. I mean – well, yes. That's right. Okay. Sorry. Your eye should gravitate towards this minus identity. And it should look at that, and you should have an overwhelming urge to carry out block elimination because a matrices is easy to invert. They don't come simpler that I. Right?

So you should have an overwhelming urge to eliminate this. By the way, if you eliminate this I, what will – if you eliminate this thing, what will happen? You'll get the short cut, and you'll go, "Oh my God. This is great. This is fantastic. That's an I. I now know that if I see an easily inverted matrix appearing as a block in the diagonal, I know what to do."

You do it, you'll form the sure compliment. That's the sure compliment. Okay? So by the way, if the I were huge and A were small, great. But of course, this is [inaudible]. So if you eliminate this block first, you're actually back where you started.

So now the trick is – so you have to actually hold off on the urge to eliminate an easily inverted block. You must hold off. This is very important, and instead, you eliminate A. Okay?

So that's the idea. And actually, let me explain the reason. The reason is this: in this application, I is small. This is – let's say A is a million by a million, but B and C are each a million by ten and ten by a million. But anyway, that's what they are. In that case, this I is ten by ten. So it doesn't really help you to eliminate this. It's this million by million guy you wanna go after. And A, for example, could be a DFD – a discrete Fourier transform. Right? Or something like that. That's the one you wanna go after.

If you eliminate that, you get this: I plus C inverse B times Y equals this. And this allows you – this, we can actually handle efficiently. Why? Because you put A inverse B here. That's fast because, somehow, you have a fast method for doing A inverse. You form the – in this case, you form the smaller one, and then you solve that. And then this is sometimes written – this has got lots of names. It's called matrix inversion lemma, but it's got lots of other names. There's – if you're British, it's called Sherman-Morrison-Woodbury, and then any subset of those, depending on if you went to, like, Oxford or Cambridge or something like that. Okay. So that's one. And it's also called – let's see. Matrix inversion lemma, Sherman-Morrison-Woodbury – there's one more. I think it's if you came from MIT, there's some name you have for it special. It doesn't matter. You'll hear another name for it. So – what's it called? Anyway – I don't know. I'll think of it. It hardly matters. The point is you'll hear lots of people talk about it, and it's – sometimes, it's just written out as this formula: A plus BC inverse is equal to this, assuming all the inverses here are – A is invertible and A plus BC is invertible. You get this formula. And you would see this – I mean, here are the types of things you should know. It's a staple in signal processing. In fact, essentially, most signal processing relies on this. This is basically the one non-trivial fact of all signal processing up to about 1975, I'd say. It's just this. Same – by the way, this is also the trick for optimization. So let me explain what it is. In that case, B and C – I mean, the most classic case is something like this: diagonal plus low rank. So here – well, I can't use lower b. So PQ transpose – there you go. So, suppose you need to solve that – something like that. So diagonal plus low rank is a beautiful example of a matrix that, if you know what you're doing, you can solve equations diagonal plus low rank so fast it's scary. But if you don't know what you're doing, it's a total disaster. So this one, you can solve. It's actually – in fact, what is the order of solving that? Order N. It's just order N. It's – there's nothing here to do. It's just order N. So if someone says, "Don't you understand? I have to solve a million by million equation?" And you go, "Yeah. I understand." And they go, "It's totally dense. Dense! A million by a million! That's like 10 to the 12 entries in my matrix." And you go, "Yeah, I know. But you can solve it in about a half a second. Maybe less." Okay? But you have to know what you're doing to do this. Okay? Nothing will recognize this. By the way, you might ask, "Is it easy to recognize a matrix, which is diagonal plus low rank?" If it were easy to recognize – I guess you know the answer, by the way. I'm asking. But if it were easy to recognize, you can imagine some processing or some higher-level thing that would look at the matrix – it could think a while. If it's a big matrix, this might be worth thinking. So it can look at it for a little while and go, "Look at that. You're lucky. It's diagonal plus low rank." So what do you think? Is it easy to determine diagonal plus low rank? No. It's a famous problem in statistics. It's in lots of other – it comes up in lots of areas. It's impossible. Okay? Let me just ask this just to see if we're on the same page here. How about this: how about alpha I plus PQ transpose? Do you think is that right? Am I asking the right thing? I think I am. All right. By the way, I'm not sure I know the answer to this one, but let's try it anyway. This might – what I'm about to say – my example might be just for symmetric, but let's just try it anyway. How about a multiple of the identity plus rank one? Can you detect that in a matrix? How?

Student:[Inaudible]. [Crosstalk]

Instructor (Stephen Boyd):[Inaudible]. The eigen values will do it. Thank you. Thank you. What are the eigen values?

Student:[Inaudible].

Instructor (Stephen Boyd):You got it. Okay. And that's if and only if. Right. So in fact, this we can determine. This we can determine. Okay? Because one eigen value will be different, and all the others will be equal to alpha. Okay?

Student:[Inaudible].

Instructor (Stephen Boyd):Correct. Right. So then we say fantastic. We have a method of recognizing a multiple identity plus low rank. Okay? And we have to calculate – all we have to do is calculate the eigen values. And what's the cost of that?

N cubed. Actually, with a multiplier, it's about five or ten times more than just solving the stupid equation. Okay.

So – all right. Well, anybody had this? I mean, the main thing here's to think. But diagonal plus low rank? Forget it. So – okay.

So – all right. So that's the picture. And these are just sort of stunning – you get stunning speed-ups if you know this material.

By the way, I should say I went over this kind of fast, and a slightly less fast description is the appendix, which you should definitely read. And we're going to assume, moving forward, that you've read it and got it.

A lot of this, by the way, you can experiment with. I mean, you can experiment with it in Matlab. You have to be careful, though, because you're kind of experimenting half with – half of it is this method, and half is what Matlab actually sort of does and stuff like that.

So – okay. So I guess we'll quit right here.

[End of Audio]

Duration: 72 minutes

ConvexOptimizationI-Lecture15

Instructor (Stephen Boyd):Great! So today we're going to start in, I guess for real, on this third part of the course, which is on algorithms for solving complex problems. The last topic, which was numerical linear algebra, which is sort of, it's just a sort of background for it. It's actually what – everything's going to be built on top of that.

Okay. So today, we'll start in on unconstrained minimization. I guess this is on your current homework as well. And actually, I'll make a couple comments about your current homework. I don't mentioning that it wouldn't take you very long, a couple of clicks at most, to actually find source code to solve the kind of problems you're looking at. I mean, particularly just, you know, for the book we have essentially all the source code online that we use. So I recommend there, so I mean you probably knew this, but I'd recommend there, actually, the code you're asked to write for the current homework is, you know, not long, ten lines, maybe. I mean conceptually it's not a big deal. It's not exactly like a heavy assignment. You should try it yourself first, just to see, but if you get stuck or something like that you type the right things into Google and two clicks away and there's all the source you want. It does pretty much what you want. Of course, actually, then you have to figure out how to adapt it to your problem and all that, but if you want to see the structure of it. So I just mention this if you get stuck with anything there you can always look there to see how to do this. You shouldn't get stuck, though. Okay. That's just a small comment about the homework. Let's look at unconstrained minimization. So here, the problem is very, very simple. Just want to minimize a non quadratic convex function smooth, that's minimized F of X. Now, of course, minimizing a quadratic is quite simple. Minimizing a quadratic you can do by solving a linear equation, because the gradient of F is F line, and so you set that equal to zero, that's a linear equation. So we're interested in minimizing non quadratic convex functions smooth.

Well, by the way, lots of methods for this, we're going to look at methods that produce a sequence of points, there're all in a domain, and the function values are going to go to this algorithm value P STAR. By the way, in the case when P STAR is minus infinity, that means this is unbounded below, you also call this a minimizing sequence. So here, I mean, in that case, of course, the Xs don't converge. Okay. And of course, to minimize a smooth convex function is the same as solving the equation, which is the degrading sequel to zero. So another way to think of this is you want to solve this set of N, non linear equations and invariables. By the way, these equations would be linear equations if F were quadratic. So this is a method for solving these systems. That's another way to think of it. Okay. Now, a couple of technical things, I will not go into too much detail about the technical stuff because in the book it's pretty, well, what's written there is actually true. But there are far more technical descriptions than that, but I don't believe they're needed. Okay.

So we'll assume the following, that we have a starting point because you have to have a point in the domain of the function. And we're going to assume that the sublevel set is closed. Okay. Now, these are sort of, I mean, mostly technical conditions. There are a few real problems where you would have things like this wrong unless you're very, very careful about it, one would be empetre, and we'll get to those later. Now, when all sublevel sets are easy to work out, but that's basically the same as saying that the Optigraph is closed and some people just call FN a closed function. That's standard notation in convex analysis to call a closed function if its Optigraph is closed. And this would be true, for example, the domain is everything, so that's one case. And another case is this, if F goes to infinity, as you approach the boundary of the domain, this has another name, it's called a barrier, but this is another condition that does the trick. Okay. So for example, one over X is going to be closed, defined on X positive, that's closed because as you approach the boundary of the domain, which is zero, from, of course, positive X, the function goes to infinity, so that's one.

Okay. So here's some examples. This one is easy, along side X of a bunch of F line functions, that's closed because, in fact, the domain of this is everything. So that's automatically closed. This function is much more interesting to understand why this has a closed sublevel sets. First of all, the domain is not everything. The domain is an open polyhedron here. So if the sets of point X for which AI transpose X is strictly less than BI. So the domain is an open polyhedron here. By the way, the condition that we know in X zero in dome F, is not trivial, in this case. Right? Because, in fact, to find such a point, you'd have to determine if a certain polyhedron has non empty interior and produce a point in it. So some of these assumptions here are not, I mean, they can have real implications, as is the case here. Now, in this case, if you approach the boundary it means you approached the boundary of the polyhedron. It means that one of these terms BI minus transpose X goes to zero. If that happens, the associated log term goes to minus infinity and with a minus sign it goes to up plus infinity. So this satisfies the second condition for sure and, therefore, this has closed sublevel sets. Okay.

By the way, it's interesting because when someone says could you minimize this, please, you could get very confused. And you could say, I can't do that yet, I'll be able to do it next week when we look at how to solve problems with constraints. You say that's not an unconstrained problem. And someone else would say, well, why is not unconstrained. You'd say because you have the constraint AI transpose X has to be less than BI. These are constraints, there's no way around that. And yet, minimizing this we're gonna discuss it now and it's going to work just fine. Okay. And the reason is – well, we'll see why. So in fact, minimizing this is an unconstrained problem even though you might imagine it has constraints. These constraints will be taken care of automatically. And if you want a rough idea now of why this appears to have constraints but, in fact, can be solved, considered as and solved by unconstrained optimization is – the reason is this: A problem like this, the constraints could never possibly be active. So, in a sense, they're really not there. That's the sense in which they're not there. You certainly couldn't minimize this and find out that AI transpose X equals BI was the essence of a constraint, for example, a linear program or anything else, is that when you solve it some of the constraints will be active. So that's the difference between here and there. But this requires actually some care and thinking so you know which is which. Okay.

So we're going to make some very strong assumptions and I should say something about that. In fact, I should say something about how this all works. So a traditional treatment of all this material is you work out various proofs of conversance of things like this. That's fine, just get a rough idea. It turns out these things are not that useful in practice. And in fact, the typical conversance will look like this. It will say if, and it'll have a long list of things, by the way, not one of which is actually generally speaking verifiable. Right. Not only that, it'll involved all sorts of constants that you don't know. We'll get to some of those constants soon, like [inaudible] constants and things like that. It will say then, and it'll have a statement that says, you know, X converges to the optimal blah, blah, blah, within and it will be some number, usually they don't even tell you the number it is, there's no bound number of sets. If there is a bound number of sets it involves all sorts of constants you don't know. Okay. So from one point of view, one could argue that such a theorem is useless in practice, and I believe that's actually a correct statement. I believe that's the case. So when you then confront people who spend, let's say, four weeks of a class, and let's say, quotes in that material, they'll something like this, they'll say, no, no, no we do that because it's a conceptually is full, and that, actually, I agree with. So it's useful to know. It's great to know something converges. Even if, when someone says, how long does it take to converge, the answer involves all sorts of numbers you don't know. So it's kind of like at least it's, I don't know, it's a nice thing to know that if you ran it forever, that it would eventually converge. That's a useful thing.

Now, in view of this being, what I believe, and a lot of other people also believe, is the role of all of these conversion theorems, it means really, since you're not gonna be really using – you might as well use a big hammer and just make the biggest assumptions you can, because these are just conceptual, in fact. Saying that is sort of the idea here. So we're gonna take basically, we gonna make the strongest possible assumptions, basically to shorten the proofs, that's why. Okay. So in that spirit, we're gonna take strong convexity. Now, strong convexity means this, it means that there's a minimal curvature, a positive minimum curvature. So we'll assume that in this case. By the way, for lots of functions that you actually have come up all the time this is false, you know. I mean, one of your answers are perfectly good example that obviously doesn't satisfy this, the curvature goes to zero. But still, we'll take this and we do just to simply the proofs because there's no reason not to, in my opinion. So you take this. This means it's a minimum curvature. Now, there's a couple of implications I'm not going to derive these, although they're quite simple, in this case, this is what it means to be convex. This says, if you look at the function at another point it's bigger or equal to, well look, this thing over here is nothing but the first order Taylor approximation and it says, basically, the function is above that.

If you add this thing here, it's actually very – in fact, it's not hard at all to prove, you write up the second order term here, and use the mean value theorem, or whatever they call it, and plug in some value in this and you'll get this. And this says, you're bigger than or equal to the first-order Taylor series, this has a minimum curvature away from it. This says, when you move away, it's not that there's a linear thing and you're above that linear thing, it says there's actually a quadratic. This is a quadratic here in Y. And it basically says, there's a quadratic that you're above. So it's a strong assumption. Which that implies all sorts of things. It implies that the sublevel set it bounded. It also implies, although it's a couple lines to derive at the following, it says that if you have a point X then F of X minus P STAR, that's the minimum value of F, it's less than one over two M times the norm of the great of F squared. Okay.

This strengthens all sorts of things. You know the optimality condition is the gradient N value of zero and that, of course, is completely consistent of this. This is now a quantitative one. And this thing here, gives you a stopping criteria. And it basically says, when the norm of the gradient is small you can stop. How small does it have to be? It depends on whatever two M. Then you might ask the question, do you generally know M? And the answer is no, you do not. So the use of this is sort of it's a bit conceptual. It means that when someone pokes into your see source and finds the stopping criterion, which is, you know, basically, if the norm of the gradient is less than some number, they say, how do you know you're close to the optimum? You'd trot out this inequality. And they'd say, yeah, but I looked at your code and I didn't see any M. And how do you know M? And that's when, I don't know, you get even more vague. And you say, well, I just made a number so small that depending on what M – anyway, you try to weasel out of it. You really can't. But the idea is that justifies this. Okay. So we'll look at descent methods after this. Actually, there's lots of descent methods, some of those interesting ones are not descent methods, but maybe, the most widely used ones for smooth problems are all descent methods. And let's look at what they are. It looks like this. And this is following now, maybe, 50 or 100, at least 50, so maybe, 70/80 years of sort of tradition and things like that.

It goes like this, you're next iterate is going to be the current point plus a direction, well, a vector, a displacement times the scale factor. Now, these are universally come to be called, I guess people call this the search direction. Now, normally if someone says a direction, they mean a normalized vector, like a unit vector, because the direction doesn't have a link associated with it. But these are generally not normalized. And then they call P the step lengths, even though it wouldn't be the length unless this thing was normalized in length, but still, that's what people call them. So that's what we'll call them. And it's generally divided into two steps. The general step goes like this. You have a point X, you calculate at that point a direction to move in, and then you calculate, it's called a line search but, in fact, it's really more a ray search, because you only look in positive P. You find a positive step size and then you update like this. Oh, by the way, usually in finding a descent direction, if the gradient of F is zero, then you terminate, right, and you don't generate. So when you determine this end direction part of that usually is something that checks the stopping criteria or something like that. So this is the general method. Now, these search directions have to make a negative enter product with the gradient. So, in another works, if you're function look like this, and the gradient looks like that, then it says that you're search direction has to be sort of in this open as space, over here. Now, by the way, when I draw this picture, there's one thing that your eye goes to immediately, and it just seems like how could you beat it. And that's the most obvious search direction there is, which is the negative gradient. We will get to the search direction in a minute.

I mean, that's sort of like why would you not go if the gradient is the direction in which locally the function is increasing as rapidly as possible, why on earth would you ever not go in the direction of the negative gradient, which is, after all, where the function is. It's the direction in which the function is decreasing as fast as possible. We'll see there's extremely good reasons for not doing this. This is often an exceedingly poor direction. But we'll get to that later. I mean, it's not a big surprise if you conceptualize it the right way. Okay, so that's in descent method. Now, in the line search you have to determine the step size and again, there's lots and lots of types of line searches. You know, hundreds and hundreds of articles, and books, and chapters, and it goes on, and on, and on. And it turns out, actually, what's kind of weird about it, is that for line searches it turns out, and it's something, actually, I frankly don't fully understand, but I accept it as an empirical thing, and that is that the actual line search matters much less than you'd ever imagine. Right. So here's some extremes. On an exact line search actually says, once you point on the direction and you have a ray, you actually sort of plot, you work out what F is along that way. In fact, for all practical purposes, you've reduced your problem to a one dimensional problem. One dimensional problems are easy. You plot it and use your eyeball. I mean, that's clear, right, it just goes like that. If it just keeps going down forever you found a direction of infinite descent.

If it points up, when T's positive it means you need to fire whoever wrote the code that generated the search direction. So basically, this is what an exact search would give you this here. And you'd think, well, look that's the point to minimizing F that's got to work well. But the other one – there are lots, and lots, and lots of line searches. But at the other extreme, there is something that is embarrassingly simple, and it goes like this, it's really, really dumb. You start with a step of one. Okay. And then, what happens is, you repeatedly multiple by a constant Beta. Beta's typically a half or something like that. So you try a step of one. And you try it. Then you check, this condition here is sometimes called the Armeho or the – I think it should be called the Armeho Goldstein, or really the Goldstein, since it is from Moscow, and it's almost certain where this came from. So this condition says this, it says when you evaluate F of X plus T Delta X, and let's actually look at a few things. Let's put Alpha equal to one. This is the derivative, is a directional derivative of the function along, in fact, you see this function right here where I erased the Alpha? This is this. And actually, by convexity, this condition here will never happen. It cannot. That's the point. This thing here is a lower bound on that. So you'll never get a decrease as good as that.

So for example, if this number here is like, you know, .6 and you try a step size of T equals one, this predicts that F will go down by .6. But it cannot go down by .6. IT can go down by .599 or something like that. It cannot go down by .601, that's for sure. Okay. So you have to degrade how much reduction you want. And in fact, what this does, is you multiple by Alpha, which is between zero and a half. We'll see why a half later. But you multiple by this, which says that you're going to accept some fraction of the predicted descent reduction and that's this. And what you do now, is you start at some T. You multiple by half, like this, until you lie below this dash one. Now, this is incredibly crude and with things like Beta equal .1, it's embarrassingly crude. It basically says, try step length of one. If that doesn't work, try .1, try .01, try .001. I mean, you can hardly said to be examining a variety of step lengths. Now, you'd imagine that, like for example, an exact line search would work way better than a line search where Beta equals .1. You can only do like three or four steps of, you know, five or ten steps of a line search of Beta equals .1, because after ten steps you're looking at ten to the minus ten steps. At that point, you might as well quit. Here's the shocking part, actually. Exact line search works a little bit better, often sometimes, in many cases, it doesn't work any better, at all. And in fact, it seems to be completely almost independent of these parameters Alpha and Beta. You can take, I don't know, Beta equals a half, Beta equals .8, .1, and it, generally speaking, doesn't make a whole lot of difference. So it's a very odd idea. But it's born out by lots and lots of empirical evidence.

So this is called backtracking, this method, and that's the idea. And what happens is, you get the first point, when you multiple by Beta, that lies below this critical value T zero. T zero is the first point where you get equality in this thing. It's where this line intersects the actually function there. Okay. So that's the picture. These are line search types. Okay. I should actually mention one important thing about line searches since you will be implementing one. By the way, I mean, as I said earlier, we're talking like six lines, if you put a lot of comments in and make this as fervors as you can. We're not talking a complicated thing. One thing very important is this, is F can have a domain that looks like that. Okay. Here. And if this is T equals one here, I mean, this is kind of obvious. By the way, this thing works perfectly provided your function returns the right thing. If you evaluate this thing and you're out of domain, if it returns plus infinity everything is cool, because – then this in equality is interpreted as false, because there's no way infinity is going to be less than anything. Right?

Now, on the hand, if you write slightly more sophisticated code, you can write one where if you ask for what F of X plus C Delta X and this is out of domain, it will actually pass back an OOD token, which means out of domain. In which case you'd have to write some logic that fixes that. Now, unfortunately, in many cases that's not going to happen. The worse part is your F function actually will, although you're outside the domain, that has to be added there special – I mean, for example, you take one over X and we're only looking at X positive. Okay. I can check if one over X – if I step out of the domain for one over X, that's this thing right here, what would happen is, it'll make sense, it'll be a negative number. It'll be a negative number, and everything's going to work, and you'll never recover from this. So this is a hint for yourselves, though. That's all right. I said more than enough on that. Okay. So now, we get to gradient descent method. It is the most obvious method, is the first thing anybody would think of. I would hope it's the first thing anyone would think of if you want to minimize a function. And it just says this, here is it, it says take the negative gradient direction. By the way, this is a classic, greedy algorithm in the computer science sense, because basically it says you want to achieve a goal, which is to minimize F, and it says, instead of considering, you know, what's going to happen in the future, you simply look at that iteration and make the maximum progress towards your goal. This is literally the greedy algorithm for minimization.

Now, once you put in that context, you can imagine now, it might not be a good idea because some greedy algorithms work, some work well, in other words, they give you actual algorithms that work very, very well, as well as any other. And some, of course, don't work at all. So we'll say this is closer to not working at all. It works in a theoretical sense. There's what you do, you chose a negative gradient, you do a line search, and the stopping criterion would typically be that the gradient is small. So the exit criterion would be actually somewhere here checked in step one. So it turns out, I'm not going to go through the details, which is in the book, it's this. It turns out that you can show this converges, I mean, with exact line search backtracking, it just works, and not only that, it converges expedientially. Okay. So that means that your sub optimality, that's the difference between F and the optimal value of F, goes down by a factor less than one at each step. Okay. Now, that's a pretty good convergence, expediential convergence, and this would be – usually you say that's great. Now, it turns out, actually – and you'd think, well, look, it doesn't get any better than that. I have a simple algorithm, it's totally obvious, makes perfect – it's what anyone would do. You have expediential convergence. You would taut that, that would be like a big fancy thing. It turns out in fact, it can work absolutely terribly, but let's look and see how that works.

And this is really sort of, you know, the one non obvious fact. You know, there's a lot of non obvious facts, but not that many in this whole course. This is one of them. That it's not obvious that the first thing that you'd ever do, why it works poorly, or that it should work poorly. But let's take a look and see. So here's just a quadratic problem. Obviously, we know what the minimum is. The minimum is zero, that's clear. But let's just run the algorithm on it. Well, when I minimize this I can work out the gradient, I can actually work out exact line search, and so on, and you can, actually, just analytically work out what the points are, what the iterates are, so they look like this. Okay. And indeed, they converge to zero geometrically, exponentially, as we were told they must.

Now, the problem here is Gamma, which by the way is the condition number of the quadratic form here, that's exactly what Gamma is. So if Gamma's bigger than one it's the condition number, if Gamma's less than one, this is one over the condition number. Okay. One of over Gamma's the condition number. Then, it says that you go like, actually, the Kappa minus one over Kappa plus one. It means something like that. Now, if you're condition number is ten, that's fine. You multiply that .90 step. If you're condition number's 100, it means that you're only making 1 percent progress and if it's 10,000 or a million, it means you're making very, very poor progress at each step. So that's – And this is, by the way, this is not a bound. This was a bound. You can always hope when you have a bound that things can be better than the bound. And in this case, they're not better, they're right on bound. And if you actually look at the iterate you sort of see what's happening. What happens is this, you're here – by the way, this is for Gamma equals ten. That's a very small condition number. You're here, and so the direction, that's the gradient and it gives you the direction of most rapid local increase. It's the most uphill direction. You go in the most downhill direction, and notice that it's pointing kind of in the wrong place. So you go along here and then, of course, you minimize along that line, that when you minimize you're orthogonal to the gradient here, and that's here. So you actually overshot.

And so it's, actually, very useful to visualize this in 3D. So you imagine the function coming up, like this, and you sort of take a ball or whatever and you roll it, it rolls down, and actually, when it hits the minimum, the minimum, by the way, is not in the valley. Everyone here, you'd know how to do this, right. I guess if you're flying an airplane about this and looked down you know exactly what to do. It says, you go down the valley and then go down the valley, because that's what this is, right. Now, instead, that's – actually, when you're at the valley it turns out you're still decreasing going this way, right. So you end up doing this. Now, this is for Gamma equals ten. So when Gamma equals, you know, 100 or 10,000 or something like that you get very, very slow conversions. Okay. Here's a non quadratic example and you can see here – well, in this case, actually, the backtracking lines search worked quite, the exactly line search, I guess, worked quite well, but just to show you a rough idea how this looks. Let's look at some bigger problems to see how this scales. Here's a problem that looks like that, and you can look at it, and these are the two things, and there's just a few things here. Actually, the first thing that's actually interesting, note that like up until about, I don't know, 100 iterations the backtracking line search did better than exact line search. So that's weird.

Actually, the reason is you don't really want to be. You can see here that, actually, being sloppy in your line search here would be an advantage. In fact, if your line search here were foreshortened, then that should be very much in your favor. In other words, if you didn't go all the way to the minimum, it would actually be way, way better. Because you'd end up closer to sort of this access here. And then on your next step, you'd shoot right in. So the first thing here is you see that there's basically not a huge difference between the backtracking and the exact line search, which is weird. The other thing is you can see it's taking, you know, it's taking hundreds of iterations. Although, I guess, it's getting a reasonable accuracy. And you can estimate C from this. And the other thing you see is, you know, roughly, this is a, I mean, just very crudely, that's like a line. And a line here, since that's a log access here, means that you're reducing your error, or you sub optimality, by kind of, roughly, very roughly speaking, a fix fraction each step. And here you can see if you do, I don't know, let's do it this way. Let's say you do 100 steps and you reduce it by a factor of ten to the four, so a ten to the four by, I guess, you have to take logs or something like that to get the right number in 100 steps but I don't know what that is, it's, anyway, can anyone do the log real quick in their head? I guess not. You can figure it out.

But then you'd say something like, C is .999, or something. That's just probably like .999 or something like that. It's easy enough to figure out what it is. And by the way, this is referred to by linear conversions if it – because it means that each iteration you improve your sub optic. You multiple you sub optimality by roughly, you know, .99 or something like that, by some number. Okay. We will get to a little more about that. Now, generalization of that is the steepest descent method. The steepest descent method says, I have a norm and I want to minimize, I want to find the steepest direction or descent. So what you do is you simply, you look and you say I want to minimize the gradient transpose times V. So V is my direction I'm going to go into. That's the directional derivative along X plus TV. That's what this is XT equals zero. But obviously, this has to be normalized so V equals one. So this would be the search direction. Now, it's usually un normalized, it's un normalized. You scale by the dual norm of the gradient. You don't have to worry too much about this. So the steepest descent is just basically so you know what it is and all that. But it's not going to be key because we're going to focus on something else. Okay.

Now, the interpretation here is that in this norm you get this tells you sort of the direction where for a unit step in that direction the first order prediction of decrease is maximized. And actually, it's a weird thing to think that it actually depends on the norm. And that's actually the key to understanding everything. You would imagine if you're on some sort of surface and you asked someone which way is the fast way to go downhill. Everyone would point in the same direction. Well, it turns out that's – I mean, obviously, if the norm's agreed upon, everybody does point in the same direction. It depends on the norm, that which is very weird. So if someone's using like a L. Because the difference is this, when you say fastest decrease of direction, you really mean, decrease in F divided by say the number of meters you move. But meters is measured in some norm. And people use different norms the denominator's going to be different and they're going to chose different directions to be the steepest descent. So this is actually – once you understand, that's actually the key to understanding everything we're going to do. Now, the convergence is something like the gradient descent method. In fact, gradient descent is steepest descent in the Euclidean norm, which is not surprising. Okay. So let's look at an example here. And once you visualize what steepest descent is you'll realize why it is when you ask two people at the same place what the fastest way to go downhill, they can point in different directions. By the way, they can't point – they actually have to point in directions that – They can only differ by less than 180 degrees, that kind of obvious. If they're pointing completely in opposite directions, there's some trouble. Right. Or it's very good or you have to minimize – well, no, they should have told you that. But they can point different directions.

Let's see what it is. If I take just a quadratic norm then it turns out the steepest descent direction is minus P inverse descent. We can actually work out what this is. You're here, and actually, I guess, first you can do the following. Let's, actually, first workout what the steepest descent is in Euclidean norm. So what you do is you say I'm going to use the linearlized model. So use the lineralized model, that's a negative gradient. And you say I'm going to move in this ball to minimize grad F transpose V, in this ball. And then answer, clearly, is right here. It's says go in that direction., which is the negative gradient. But now, suppose, in fact, that you measure your displacement using this quadratic norm this is the ball. Then it says, go as far as you can in this direction in this ball. And what happens is, it's here, it's been rotated over to this direction. It's been skewed over here. So you can see, these are, obviously, not the same direction, and you've actually twisted the thing. Now, in fact, you can't twist it more than 180 degrees because that's a positive definite matrix and you multiple a gradient of one vector by a positive definite matrix, you don't rotate it more than 180 degrees. In fact, that's 90. Well, whatever, this 180. That is exactly what – actually 90, sorry. That is exactly what a positive definite matrix is. So you've twisted it over like this. It gets interesting, too, if you wan to do steepest descent in the L one norm because you'd draw a ball like this and take the gradient like this. By the way, what would be the steepest descent direction in the L one norm if this were the gradient, if that were the negative gradient, what would be the steepest descent direction?

Well, what's the farthest – it's weird, it's curved, that's kind of a weird thing. All right. Imagining that I had drawn that arrow straight, what would be the steepest descent direction in L one? What would it be? Well, you go as far as you can in the direction and so it would actually be this point here, and that would be your direction. And you can kind of make a guess about steepest descent in L one. Go ahead and make a guess as to what it –it's always a long unit vector. Or can always be chosen to be a long unit vector. If you're going along a unit vector it actually has a very beautiful interpretation. It basically says you're updating one component of the variable. So the deepest descent in L one at each step you optimize over one component of the variable. Okay. So that's about it. Okay. That's not actually – our main interest is not that. It actually has to do with the choice of the norm for steepest descent and so this will illustrate this here. Now, the first thing we'll take – now, by the way, these are sublevels of some function here, some function we just looked at, like this. And the point is if you look at his function what you see, it's very rough, but this is the idea. The function is sort of elongated, it's not spherical. Right. It's kind of fatter than it is tall, roughly. Okay. That's the main thing here.

If you chose a norm that is sort of aligned with the gross geometry of the sublevel set, an interesting thing happens, you end up, actually, making very fast progress towards the center, okay, because you actually end up going in the right direction. If you take a norm, which actually is whose unit ball is sort of not aligned the same way the gross geometry of the sublevel set is, it actually amplifies this effect of osculation. Okay. So the rough idea is something like this. There's another way, also, to understand this, which I'll get to in a minute. Basically, to understand something like this, if you really want the norm that you use, if you use steepest descent you want the norm to sort of be consistent with the geometry of your sublevel sets. Okay. So if you're problem is the sublevel sets, you know, we're sort of more or less spherical, you know, like that, right. By the way, if the sublevels set were spherical, how would gradient descent work, or nearly spherical? How would gradient method.

Student:[Inaudible].

Instructor (Stephen Boyd):Actually, if it were perfectly spherical it would work flawlessly. It would say that's the negative gradient, it would go in that direction. Negative gradient would point, if not to the center, to the optimizer, it would point very near it. You'd minimize it and you'd end up over here. So actually, the gradient works perfectly when your function is, and I'll try to say this right, when it's isotropic or something like that when it kind of extends, when it looks about the same in all directions. It is kind of spherical or in all directions it looks about the same. Gradient descent works absolutely horribly when the sublevel sets look like this. Okay. That's exactly when you get this sort of story. Okay.

Now, when you do steepest descent in a norm, it's the same as changing concordance and then applying gradient descent. So what you want to do is you want to change concordance to make things that look like this look like that. So the whole key, in fact, is choosing the norm. That's the key to everything. Okay. So the norm you do steepest descent in is very, very important. Okay. Now, by the way, this already, if you just incorporate this amount, you already know a lot. If you really – you actually know a lot more than a lot of people who do optimization. If you just understand just what I just said. So for example, suppose you know something about the function you're going to minimize. Right. Suppose you know you have samples of it or something like that. You can do all sorts of cool stuff. You could like fit a quadratic. Suppose you already evaluated the function like a couple hundred points, you can fit a quadratic to it, or do whatever you like.

You fit a quadratic and a quadratic would sort of tell you which direction the function is sort of, you know, long in, and where it's got steep canyons. You would then use that quadratic to change concordance or if you like, you can use that metric to do steepest descent. Everybody see what I'm saying? That would work extremely well. I mean, that would improve steepest descent tremendously. Okay. Now there's a very obvious choice, though. If you want to get to, let's see, I want to motivate this, so I motivate it this way. Let's suppose you're near the bottom of a function, you're near the minimizer. So here you are. It's smooth. Actually, what does the function look like near the minimizer? What's an approximation of F near the minimizer? So well, it's equal to F of X star, right, plus grad F of X star, transpose X minus X star, right. What's that?

Student:[Inaudible].

Instructor (Stephen Boyd):That's zero. Okay. So we go to our next string, which is one half, X minus X star, transpose, you know, the Hessian, X minus X star. Right? So that's our next string.

Now, near the optimum point, the optimal point, it looks quadratic. And so what does the sublevels of F look like, approximately, near the minimum? They're ellipsoids. And they're ellipsoids determined by the Hessian. So they look like this. You know, they're ellipsoids. Hey, that's funny. That's exactly what we want because it tells you what the shape looks like. And it basically says, if for some reason you knew that number, of course that would be very – I'm sorry, that matrix, that would be very interesting that you knew that because the only way to find that, I guess, would be to find X star first and evaluate it, but suppose anything intelligent you could guess about this Hessian, it's going to tell you something about the shape of the sublevel sets, certainly in the end game when you get close. Okay. So then you can imagine, I'll do steepest descent, but I'll do it in the norm induced by the Hessian. Okay. And that brings us, that's very famous, that's called Newton's method, and that's our next topic. Okay. So let's look at the Newton step. So the Newton step is this. It's actually, in fact, a – it's most way interpreted, it's minus the Hessian inverse times the gradient. That's all. And you can say lots of things about it. So here's one. One way to motivate what the Newton step is this, is you take your function and you develop a second order approximation, that's this. And then it says, I'll minimize this, instead. If you minimize this, I mean, you just take the gradient with respect to V equal zero and you'll get V equals the Newton step here. So this minimizes that. That's one way to think of it. And, in fact, that's a very good way to think of Newton's method. Newton's method will use the Newton step.

And it basically says, you take a function at each point, you develop, not a first order approximation, but a second order, and you minimize that, and then you update. So that's this. Another way to say it is this, you want to solve, if you're at the point X you would like to find a V for which the gradient of F, evaluated X plus V, that's where you're going to step. That should be zero. But the problem is the gradient nonlinear function, you don't really know what gradient of X plus V is. Okay. All right, so what you do is you say, well look, this nonlinear function is about equal to – I'll replace it with this linearized approximation, which is the gradient where you are plus the derivative of the gradient, but that's nothing but the Hessian. So this is that. And then you set that equal to zero. And if you solve this equation, of course, you get V equals minus Hessian inverse gradient. You get that.

And the two pictures correspond to something like this down here. So in the first case, here's F, our function, and here's our point X. So in the first interpretation it says develop not a first order Taylor series approximation, but a two term Taylor approximation. By the way, when you carry out two term Taylor approximation, convexity is preserved. If your originally function is convex, so is your two term Taylor's expansion, obviously, because the Hessian is positive semi-definite. And it says minimize that and that would give you this one, this point here. Notice that you know, at that point already you can see that F hat and F are not the same. By the way, what will happen in the next step here? If that's our Newton step and I carry out this again, I fit a quadratic to this point and do it again. What's going to happen? It's actually going to be much better this time. And when I start getting near this bottom point, these quadratics are going to extremely good approximations and the improvement you're going to make each step is going to be dramatic. Okay. Another way to see it is this, here's F prime and we want to solve F prime minus zero, that's our optimally. So here's my – and it's convex, which means F prime is increasing. So F prime is increasing. That's F prime. I'm here and evaluate the derivative of prime. That's the second derivative. Okay. And I form this dash blind, which is, in fact, this thing. Okay. Which is here is very simple. And I take the zero of that, that's that. By the way, what happens on the next step?

When this one is extremely easy to see what's going to happen. You're going to put a line through here. This is not linear here or F line, but it's very close and the next iterate is going to be somewhere in here. When you get in there, the iterates are going to be unbelievably accurate and you're going to get very fast conversions. Okay. So this is sort of the picture. So by the way, people call methods, where you update sort of this matrix here, they call these a beautiful term for this, although it means, let me be a little more specific, it's a variable metric method. So metric here refers to how you describe distances and things like that. So a variable metric method is a method that varies. It's essentially, the norm being used in steepest descent at each step and it adjusts it to the geometry of the particular problem instance. That's what a variable method metric is. So although you wouldn't call a Newton method a variable method, it is one in that broader sense. Okay. So you can think of the third interpretation is this, is that the Newton step is steepest descent direction in the local Hessian norm, something like that. So that's what it is. That's the picture. Okay.

Let's look at the Newton decrement. Now, the Newton decrement is actually, it's – the Newton decrement square, it's defined as this. It's actually the norm, [inaudible] is the norm of the gradient but in the metric induced by the Hessian. So that's what this is. That's that. And it's a measure of the proximity of the X to X star. And it's actually a good one and we'll see some very good properties it in a minute. And it allows you to make a very good approximation of how suboptimal you are. So one half [inaudible] of squared is a very good approximation of how suboptimal you are. And it's basically the norm of the Newton step and the quadratic Hessian. It's also the direction of the derivative in the Newton direction so it turns out you actually have to calculate this whether you like it or not because that's the thing that you used in the line search. That's the RB hole. This is the quantity that appears on the right hand side of the RB hole line search anyway. And here's the really cool part about it. It's F line invariant. And let me say a little bit about that with Newton's method. We haven't talked about Newton's method, but I'll say something about that in a minute, but let's – yeah, okay. Let me get to Newton's method and then I'll come back to this.

So Newton's method, very, very simple, looks like this. You're given a starting point and a tolerance, or something like that. You compute the Newton step in decrement. You know, obviously, here if land to square is small enough you quit. That's fine. Otherwise, you do a line search and then you update. That's Newton's method. All right. Now, what we'll find out is that this works unbelievably well. Shockingly well. We'll take a look at it. We'll see in a minute. But there are many reasons one, I think probably the best zero order description of why it works is F line invariance. So let me explain what that is. If you want to minimize F of X, I'll try to get the notation here right. Okay. I could change concordance and let TY equal X, where T in a nonsingular matrix, and I could, just as well, minimize, you know, F of TY over Y. Obviously, I could do that, a change of concordance. Okay. But there's a little bit of – and then you ask what do I apply if I change concordance and then applied gradient method, for example, you know, do you get the same thing. And the answer is, no you don't. And in fact, what happens is in a gradient method when you take, if you call this thing F tilde of Y, then grad F tilde of Y is actually T transpose time F of TY. Okay. So you get that. What have I done there? So you actually, it changed concordance and that means, actually, if you change concordance and then apply gradient method, these don't commute. It's not the same as applying gradient and changing concordance. They don't commute. Well, in fact, that's the message of everything we just looked at, if you change concordance or change the metric the gradient method changes.

Now, you can look at it as an optimist and say, hey, that's cool. If I only get the right concordance gradient method is going to work really well. In fact, we know what the right concordances are geometrically, we want change concordance so the function looks isotropic, it looks round. That's what we want, we want the function to look about the same in each direction. We don't' want steep canyons. We don't want stuff like that. If it kind of looks like the same in all directions, gradient descent is going to work really well in that metric. So that's the idea. Okay. Now, the truth is, there's a lot more bad choices of concordance than there are good ones. So with all due respect to the optimism express by saying there is a choice of concordance, which makes gradient method work well, which is a true statement. There's a lot of ones that don't. Another way to say it is this, if by any chance you're problem is solving well with gradient descent, and by the way, there are plenty of problems that solve well with gradient descent. Obviously, these are the ones, which, for various reasons, are round, or they're not too un round. Right. They're the ones that where they kind of about – there's lots of cases where gradient method just works. For super large scaled problems, you know, it's about one of your only choices. So you know, good for you, if a gradient method works. Right. But in fact, they generally don't work for smaller problems and I can always just change concordance. I can even take T diagonal. And I guaranteed you if your gradient method is working and I change concordance, I mean, I can just scale your variables, and I will bring your gradient method to a halt. Of course, in theory, it will convert. But I can make your C .99999999 and you're going to have a lot of iterations. Okay.

Now we get to Newton's method. Newton's method, actually, you get a commutative diagram. So Newton's method and affine changes of concordance they commute. So in other words, if I want to minimize F of X and I change concordance and I do something like I minimize F of TY here. I will get a sequence of Newton iterates. Okay. So this is Newton, this is change of concordance, right. I will apply Newton and I'll get, you know, X one, X two, you know, and it's going to converge. And here, I'll get Y one, Y two, and so on, like that. Okay. Now, I guess, if you have any aesthetic sense at all or even a small training in mathematics you have an overwhelming urge now to – you're hands are shaking because you want to fill this in. Its' just a – anyway, so you want to fill – and that's called a commutative diagram. And it basically means, it's cool. It says, basically, this basically was called a commutative diagram because it's basically Newton's method and changes of concordance commute. And in this case it's correct. By the way, if I replace these with gradient method, it's completely wrong. It does not commute at all. These are totally different sequences. And from a practical point of view, this is very, very important because it means, for example, these X's could converge basically to a good enough solution in ten steps and this could take ten million. Okay. So change of concordance and gradient method do not commute, very important. Changes of concordance and Newton method, what did I just say? It was wrong. That's the problem with these things being on TV. All right, here. I'll say it again and this time I'll say it slower and it will come out right. I think I said – did I say it wrong?

Student:[Inaudible].

Instructor (Stephen Boyd):I did. Oh, I've just confused myself. Okay. That's fine. I'll go slow now. Changes of concordance and Newton's method do commute. I was right. So they do commute. And actually, that means all sorts of cool stuff. It means, I can tell you what it means. It means to first order scaling is a non issue in Newton's method. In fact, it's actually only numerical that it's an issue. And let me explain that. In gradient method, if the condition number is 1,000 of the sublevels sets or whatever it is, you're in deep trouble because that means you will be making .1 percent progress at each step. That's what it means. That's what it means. So in that sense, you know, a cap of 1,000 is already bad for gradient method. Now, if you take Newton's method and you do everything exact arithmetic, it's just no problem, it's identical. There is a second order effect. The second order effect actually has to do with numerical and your algebra and actually solving HV equals minus G, where H has high condition number. But a condition number of 1,000 for a positive definite thing is a joke. It's basically zero for all practical purposes. That's an extremely well conditioned positive definite matrix from the point of view of the numerical analyst. From the point of view of the gradient method, it's basically slows the gradient method to useless.

I mean, 1,000, I was drawing pictures here with ten. And you could actually visually see the ridiculous osculation. Imagine what it is for 1,000. Okay. And in a lot of dimensions where you can't visualize it. Anyway, it will be very, very slow. There'll be lots of osculation and stuff like that. And it'll basically be unusable but for Newton's method, no difference, absolutely no difference, it will just run perfectly. Okay. So that's the idea. So let me tell you a little bit about the convergence analysis in Newton's method. So Newton's method, first we'll look at the classical one. This is Kontorovich. You'll never guess where he was a professor. I figure your first guess is right, actually. So this is, although I'm not pronouncing it right, but I'm sure there's people here who speak Russian, you know, can tell me how. But that's how I pronounce it and I don't know. It's like Gaous or – anyway. Okay. So here are the assumptions. We'll make F as strongly convex on S with constant M, and the Hessian is going to be [inaudible] continuous. Now, by the way, this is essentially a constraint on the third derivative of F. Right. Because basically we're saying how fast can the derivative change. You can interpret L as a limit on the third derivative of F. Now, let me tell you why we don't do it that way. Because what is the third derivative of a function from RN to R? Here's a hint. Only people in mechanical engineering and physics would not be bothered by what it is.

Student:[Inaudible].

Instructor (Stephen Boyd):Thank you. That's a tensor. It's a tri linear form, if you're in math, or it's a tensor. It's got three N to C's and all that kind of stuff. It's not fun. I guess, actually, if you're fully trained in all these things, I think it doesn't scare you at all. It's no problem. In which case, all I have to say is good for you.

So anyway, it's easy. All we need is a bound on it so we'll just put it this way. By the way, before we get into this let me ask you some questions. How well does Newton's method work on quadratic functions?

Student:[Inaudible].

Instructor (Stephen Boyd):One step! It works super well. Why? Because Newton's method does this, it takes a point and develops a quadratic model of your function. Now, in this case it happens to be your function because your function is quadratic. Okay. It them minimizes that quadratic function, which unbeknownst to it is your function. And then steps to what's the minimum and asks again, say, hand me the gradient and the Hessian, and the gradient is zero, and then it actually sends a message back and says, don't even bother with the Hessian. So it's one step. Okay.

So basically, when do you think Newton's method's going to work well? Well, it should work well when the Hessian is slowly changing. So if the third derivative is small Newton's method should work really well. Is everybody kind of seeing the idea here? Because, basically, Newton's method is you're basically, let's see, at each step you're actually minimizing a quadratic model. If your quadratic model is really good, even for big displacements, Newton's method is going to work really well. So basically, small third derivative is what makes Newton's method work. Okay.

So I'm just getting you all prepped for the idea that L, the ellipsoid's constant, is going to play a role in Newton's method. And the ellipsoid's method on the Hessian is indeed a bound on the third derivative. And we know, at least in one extreme case, if L is zero then Newton's method works very well. One step is quadratic. Okay. All right.

Now, here's how Newton's method, the classical analysis works. It's going to work like this. There's going to be two regions of sort of convergence and it's going to work like this. If the norm is bigger than some number, and that's some positive number, then what's going to happen is F, you're going to be guaranteed at each Newton step to decrease F by a constant, just an absolute constant. Okay. Now, by the way, that means that this step, if F does have a minimum, this step cannot be executed more than some number for times. Because basically, it says you guarantee a decrease of, you know, whatever it is, .26, at each step. But that means if you make case steps you have reduced F by .26K so you can't go too – if you went infinitely off in doing this you'd have a minimizing sequence going to, it would unbound up. Okay. So that's the picture. Now, so eventually the gradient gets smaller than something. Now, what if the gradient is smaller than some number [inaudible] you have the following. You have to look at this very, very carefully. It's got a constant L over two M square, don't worry about that, but look at this whole thing has measure of error. That's a measure of – well, it is a scaled residual, and this says that the residual is less than the residual squared at each step. Okay.

Now, by the way, once this number – let's suppose, at some point, this number becomes one half, just for example. So this number gets down to a half. I says on the next step it's less than a quarter. And on the next step it's less than a 16th, and then 1/256th, and then 1/256th squared. Everybody see? So and notice the convergence then is extremely fast. You're basically, once this number – it basically says the error is squared. It's all very rough, but it says, basically, that the error gets squared every step. Once that error is small, that's a huge reduction in the error. Compare that to saying the error gets multiplied by .99 at each step, huge difference. Okay. So let's look at how this works. Now, when you're in the so called damped Newton phase, damped Newton means, well, we'll see it, it means you actually require backtracking steps. It means that a full Newton step you actually have to backtrack from T equals one to whatever, something. And the function value decreases, it has to decrease by a least Gamma. Now, as I said, if the function is bounded below, this has to end after, at most, this number of iterations. That's obvious because at each step – otherwise, if you ran more than that number of iterations, you've really reduced the value of F to blow it's minimum value, which is not possible. So that bounds the number of steps you take in this stage. Once you get into the so called quadratic convergence stage, it basically says the following. It says that this thing, here, is less than one half two to the L minus K. So that's how that works. So once you've gotten to that point. So at this point, you're actually, I mean, actually, many people say this many, many ways, one way is to say the error is squared at each step, not quite true. Another way is to say the number of actuate digits doubles every step, because that's what this says. Well, it says it in bits, actually. And so the number of accurate digits doubles.

Of course, you know, most normal people use doubles and floats, a tripling floating point. So this is amazing, but it basically says that you know, you can't execute this more than about four of five times before you get the double precisions stationery. At which point, normal people say you're done, you're actually done. Okay. So if you work out the global complexity analysis it works like this. It says that the number of iterations until you get less than S one, is less than this, that's the initial phase. And look, it depends on how bad your initial point is. So if your initial point, I mean, basically, actually, it depends on how suboptimal your initial point is here. And then there's the second one. This is the quadratic term. This is log, log, log, F one zero over F one. F one zero's one of the constant. So this is it. Now, I can, let me explain a little bit. Actually, I go around and give talks often about things like this. And what I do, actually, when I do that, is I actually give talks and I replace this with the number five. I'm just bating people. So it's for me to torture complexity theorists. So I wait, I wait, I just keep going with the five, I'll even stare right at somebody I know, just kind of look at them right in the eye and I'll say plus five. Just baiting them, baiting them, and finally, someone takes the bait and they'll say, excuse me. And I'll say, oh, yes, do you have a question?

And they'll say, yeah, you know, you can't solve that problem exactly so what happened to the accuracy Elvin? Huh? I don't see Elvin in there. And I say, oh, sorry, this symbol five, that's how I write log, log one over Elvin. So from now on whenever you see the number five, it actually means log, log Elvin zero over Elvin. Is that okay? Because that's how normal people write this expression, is five. Okay. So it works pretty well, actually. They do fall for it pretty often, actually. All right. So the interesting thing about this, actually, is it says that to a normal person it basically says that, actually – let me ask you this. How does the number of Newton steps required, and I'm talking by the theory, even by the theory, and by the way, in practices it's even stronger, how does the number of steps of Newton iterations required change with the accuracy required? Though, if you're in a court under oath, what would you say? You would say, very, very slowly, log, log one over Elvin would be the answer. But any other place, what would you say? You'd say, not at all. And someone would say, really? The cost of computing a super accurate answer is the same like a crude, sloppy, terrible answer? By the way, for gradient method, those two numbers are grossly different, right. Because you paid for gradient and it's hard work. You keep multiplying by some number that's .99 and you have to do it a lot more times before you drive something to an error that likes one E minus eight or one

E minus 12 or whatever it is, right? It's going to cost a lot. For Newton's method, it's basically costless. Everybody see what I'm saying? Okay.

So that's why, by the way, people don't worry a lot in practice about what's the stopping criterion for Newton's method. Because roughly speaking it's this, once you're in to the quadratic, because the cost of calculating a super accurate solution, is essentially like one or two steps more than calculating an okay solution. So it just doesn't matter. That's one way to say it. Okay. Okay. Now there's two minor problems with this classical conversance analysis and I'll say what they are. The first is that, obviously, this goes back to my comments before, obviously, in any particular problem, you have absolutely no idea what L is. Number one, you have absolutely idea what M is. Now, the other things appearing in here, like in Gamma, you have to have Alpha and Beta, these are the line search parameters. You do know what those are because they're in your code. Okay. They're in your – you just look through your code and you'll find them. But you don't know what M is, you don't know what L is. That is ridiculous. Right? So the point there is this is still useful because you will observe sort of a damped phase and then a quadratic convergent phase. So let's look at some examples. Here's an example of Newton's method for that problem before. And you want to notice two things in this example. You want to notice actually the access here. And you want to notice the access here. And by the way, here you see, let me show you what you see. You can find iterations. So the point is but here you have this region something like that, you would argue maybe here, it's sort of the damped phase.

You have to see this. You have to see this thing roll over for Newton method. You'll implement one this week. Actually, in the next two days, right, it's Tuesday. So if you don't see this thing rolling over, you're not achieving quadratic convergence. That's what it means. Okay. It is possible, actually, that quadratic can have this look kind of flat and then roll over later. But we wouldn't assign something like that to you because it would give you the wrong message. Generally speaking, if you see it flat, by the way, if you're plot looks like this, yeah, then there's something really seriously wrong. Okay. But if you don't see that, you're not doing Newton's method. Also, these are about the right numbers here. And you might ask how does that scale and all that kind of stuff, and you would see something like this. This is an example of our 100. Again, you can note this axis and again, exact backtracking line search saves you one step or something like this. And you want o know how this scale. Here's an example in our 10,000. And it's very close, I think, to what you might be doing, anyway. So here, actually, you can see it quite clearly. So this is our 10,000, you're minimizing some smooth convex function. And, in fact, you might argue that quadratic convergence goes into effect somewhere around here. Something like that. I mean, who knows, but that's roughly what it is. This is what you see.

So you might have like 13 steps and then, once you hit quadratic you'd go six more and it's all over. And you get something like this. This is our 10,000. That's a big place, by the way. You might ask what this look like at our ten million, and the answer is it looks exactly like this.

Student:[Inaudible] computing the inverse [inaudible] compared to the cost of the actual steps you have to do after [inaudible].

Instructor (Stephen Boyd):That's a very good point. So let me actually say something about that. Because that's a great point. Also, I'll short circuit a whole section of what you'd hear in other classes. Okay. So you should notice that this axis, it is completely unfair to compare this axis to the one in the gradient method, for precisely that point. Each step here requires solving HV equals minus G. That's how you calculate a Newton step. Okay. So in a dense case, the cost of that is going to be N cubed. The cost of evaluating the gradient and all that, we could maybe make it N. I mean, it depends on what the problem is and all that. But let's just say it's N or something to keep it simple. It'd probably be more like N squared. But let's make it N. So then you actually have to take that into account. At which point, you'll actually still find the following. If you need high accuracy, nothing will beat it. Nothing will beat Newton. So that's it. Now, by the way, there's a whole family of methods in between gradient and Newton. And these came into being in the '60s and they came into being because solving, for example, there are even very famous people who said things like, no one will ever have to solve, it wouldn't even make any sense, to solve 100 equations with 100 variables. That just doesn't make any sense. There's never a time when 100 things could depend on another 100 things. And besides, there'd be 10,000 co efficiency in there. You see what I'm saying here?

So this would just be inappropriate to solve AX equals B if AX is B. I mean, we're talking iron corps memory days, which means nothing to any of you. And by the way, it means nothing to me, too, just for the record. But still, you know, this is a day when it was a big – N cubed was a very scary thing when N was 100. Okay. For you, N cubed doesn't start getting scary right now this year until N equals 2,000 and it's not even that scary. Right? It's not that big a deal to make it go a bit higher. It'll start getting scary around 5,000, 10,000, that could get scary. But that's what 100 looked like before. So in those days they worked out all sorts of methods where to avoid getting out of solving H here, HV equals minus G, they avoided the N cubed trick. And they did things like they would update H by eristic and stuff like that at each step and so on, and there's lots of methods. By the way, some of these methods are quite good and they actually do have their uses. We'll look at them next quarter. And these are called quasi Newton methods. They have all sorts of names like this.

So I have a couple comments about it. So the first is this, in terms of the cost of Newton's method. If you don't know how to solve linear equations, and I'm sorry to report that many, many people do not know, then they overestimate what it costs to solve HV equals G, this thing, HV equals minus G. Okay. So when you're more sophisticated, like you are now, after last week, and H is sparse, you know banded plus low rank, if it's got Copeland structure, anything like that, you will realize all of those tricks can be brought to the bear here. So I think, one of the things that you have to remember is that in many problems, when you compute the Newton step, there's lots of structure lying around in real problems, lots and lots. And it it's exploited, which is to say that you do your linear algebra intelligently, as opposed to stupidly, you'll find, often, that the cost of Newton, that Newton's method is entirely feasible even in things like signal processing problems, image processing problems, where the number variables in a million or more, you can find you can do Newton's method. Obviously, you couldn't even, remotely, write down the Hessian and store a Hessian in a single processing, memory-processing problem or a large machine-learning problem, you couldn't do it. But you know what you're doing, you can actually solve these equations, shockingly. So that's a longwinded answer to this.

So it's not 1963, in which case the rest of the quarter would be methods for avoiding using Newton's method at all costs. It's not. So things have changed and now, Newton's method is just not a big, it's just not a big deal. And especially, coupled with if you know how to solve linear equations intelligently. And a lot of times, the cost of Newton method drops right down to the quasi Newton methods from the '50s and '60s. So okay. Well, we'll quit here and have fun, because you're going to do all of this.

[End of Audio]

Duration: 77 minutes

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

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

ConvexOptimizationI-Lecture18

Instructor (Stephen Boyd):Right this far – we're on? So we didn't get very far. We just started talking about the basic ideas of barrier methods. Now, but we'll really do it today, so we'll look at how it all works.

Okay, so the idea actually is very simple. If you can go down to the pad here. You start by rewriting the problem. You take the constraints, and you simply put them this way. It's f0 plus – there we go. Plus, and then this. I minus is the indicator set of negative reels.

And so i is this function that's zero, and then goes to plus 8 if you're above zero. And this is, of course, this encodes the semantics of a hard constrain. It basically says you have no irritation if it's less or equal to zero, and if it's positive, you're infinitely irritated, meaning, that that x is completely unacceptable.

So this – I mean, we can then say look it's just an equality constrained problem here, but if, of course, this is just rewriting things. And it doesn't really help anything, because this function, it is true the only constraint now is an equality constraint, but it's function is, of course, is highly non-differentiable and because we've put this here. That's exactly the point.

Okay, so what we're going to do is we're basically going to replace this indicator function with an approximation of the indicator function that's smooth. So and this is being many ways to talk – a lot of people talk about this. It's something like this. When I'm making fun of people who propose things like this I usually say this is the same as getting out some sandpaper and saying you don't like this corner here so you sand off the corner or something like that.

So we talked about this at the end of last lecture. So a very common approximation is the log barrier. So the log barrier goes like this. You have the sum of the logs of the negatives f. Obviously, if f becomes zero or positive this goes to minus 8, and with a minus sign, it goes to plus 8. It's a barrier. And you put a one over t in front. T is going to be a parameter that controls, you know, how close this thing approximates this indicator function.

And it's important to understand the tradeoff here. The tradeoff is this. The larger t is, the closer this function looks to that. That's clear. But that's both good and bad. It means that by solving this problem you get – well, we hope it means. We'll see soon if it's true that you're getting a closer and closer approximation of this problem. It certainly means that. But on the other hand, the larger t is and the more this function sort of looks like this one, where it's just basically this indicator function with the corner sanded down. That's going to mean this is a harder problem solve using Newton's method.

Okay? So that's going to be the trade off here. And we'll see how that works around it. So that's the idea – is that we're going to solve this using Newton's method. By the way, there's nothing wrong with solving this with Newton's method. This we know how to solve that. Any value of t, you have a feasible point. We can actually, a strictly feasible point. Newton's method will work perfectly on this. It's just no problem.

Okay. So this function here, which is the sum of the minus log of the minus constraint functions is called the logarithmic barrier function associated with the constraints. So that's the log barrier. And this minus log minus f. That's convex function by one of these composition rules, because minus log minus something is an increasing function and so on. Increasing in convex. And then f is convex as well, of course, so that's convex. It's a log barrier, and you can get a rough idea of what it is. For example, another way to say it is this. It's the log; it's minus the log of the product of the minus fi's.

So these are slacks, because that's literally how much. That's how much extra you have in the inequality fi of f(x) < 0. So minus fi is the slack. And this is negative log, the product of the slacks. So that'll come up a bunch of times.

Okay. For the record, we'll work out what the derivatives are. So the gradient is simply this – because that's just a composition rule for the log. And the Hessian looks something like this. It's got these rank one terms here, and then it's got these things with the scaled version of the Hessians of the individual functions, okay? So that's the – these are the gradient Hessian. We'll come back to this. We'll actually need the gradient very soon.

All right. Now, when you minimize f0 + 1/t fi(x). It's the same as minimizing t(f) 0 + fi(x). It's the same thing. And when you minimize this, we'll actually refer to the minimum of this as x *, the argent of x *(t). So that's going to be the minimizer of this function, subject to that. And we'll call that x(r)(t).

We're just going to assume here that there's a unique minimum, and we'll call it x *(t). There – well, we'll get into that later. Actually the sublevel sets are bound to this unique minimum, but we'll just leave, we'll just say this is the minimzer is x * (t).

This is a function of t, of course, and it traces out a curve. A path. And that's called the central path of this optimization problem. So you call that the central path, and the intuition is something like this. As t goes to infinity, you would sort of guess that x*(t) is going to go to an optimal point of the original optimization problem, because as x*(t) goes, as t goes to 8, the difference between this thing and this. I have to include the minus sign there. The difference between this and this basically goes away. Because as t goes, certainly element wise, point wise – well except if you're at zero. But element wise it's fair to say that this thing converges to that as t goes to 8.

So you sort of guess that this is the case. And here's an example for an LP. This is an LP and r2, of course, so it's kind of silly. But here's you've got some inequalities. I guess one, two, three, four, five, six inequalities like that. And then these dash curves show you the sublevel sets of the barrier function. That's this thing.

Like that. That's the sublevels of the barrier function. And you can see the one in the middle, that's with t = 0. That's, you minimize the log barrier. By the way, that's the same as maximizing the slacks. The products of the slacks, or the geometric mean of the slacks. So that's actually called the analytic center of this polyhedron. And that's right there. And then as you increase t now, what happens in you're incentivized to – well, let's talk about what phi does. Phi is a barrier and what this does, it wants to keep you away from the boundaries. Because if you get near a boundary, one of the fi's is going to be negative, but near zero.

That means one of the minus fi's is going to be small. One of the log of that is going to be big and negative. And with the minus sign, it's going to be big and positive. So this barrier function, if you can visualize it goes something like this. It's got its minimum there. And then I guess it curves smoothly up. It's an analytic function. It curves smoothly up. And then when it hits these walls it goes to infinity. So you should visualize that here.

Now when you crank up t, what happens is – by the way, you'll never find a point that is outside the interior of the feasible set. Because this thing, the domain of this is the interior of the feasible set. So no matter how high t goes, you'll always have a point that is interior. And by the way, that's why one of the names for these things is interior point methods.

We'll see later actually that modern interior point methods don't work this way anymore, but still the name has stuck and that's why they are called interior point methods. Because every point you produce is in the interior. That used to be the case, anyway.

Okay, so what happens is you crank up t, and you're going to move more and more towards. If t is this c, you're going to move. This is the curve, this is the central path, and you can see that as t goes to 8 you're going to approach this optimal point. Okay? So that's the picture. That's how this works. By the way, so far you should be deeply skeptical of all this, because well, for the reasons I talked about last time, but I'll talk about them again.

Let's see, the other thing I should mention is this is a sort of a famous method. There's a general class of methods that are worked this way and they're called homotopy methods – like this. And I can sort of explain a little bit about how these work.

A homotopy method works like this. You have a problem that you want to solve but it's hard. And you don't know how to solve it or something like that. What you do is you make a parameter. You then introduce a parameterized family of problems. Parameterized by, let's say, a parameter t – let's say. And you arrange for the problem to be easily solved when t has some value like zero. In this case, it's just – you've already written code to do that like two homework's ago or something like that. So it's simple to find this analytic center.

Then what happens as you crank the parameter – the problem you're solving is going to transform into the one you want to solve. So in this case that happens, roughly speaking, as t goes to 8. So the homotopy method works like this. You have a parameter value. You solve the problem. Then you crank up the parameter a little bit and solve it again, starting from where you started before. Hopefully that works or whatever. If it doesn't work, maybe you were too aggressive in your t update or something like that and you back off. I mean, you can think of all sorts of methods for this.

By the way, these methods are very widely used. I mean, all over the place – so that's what this is. I'll come back to that. Okay.

Now let's actually do some. Let's see what happens on the central path. Well, if you minimize – f you solve this problem, then it means that the gradient of this plus a transpose nu is zero. That's the dual residual being zero, and of course Ax = b. So it means that the optimality condition for minimizing this barrier-augmented problem is this. You have to have the gradient t grad f0 +, and that's the gradient of the barrier, plus a transposed w = 0 and Ax= b. So that's the optimality condition.

Now if you stare at this long enough you will look at it and realize you've stuff like this before. And in fact, what you saw before is this. We can take this thing and we can divide by t. That's the first thing we'll do. Divide by t like that. And then you stare at this long enough and you realize these are just like positive numbers. So I'm going to call those just lambda i. And then you're going to look at this and say wait a minute, this is grad f0 plus sum lambda I grad f I plus a transposed something equals zero. And you realize like wow, look at this. If you define lambda i this way, this equality means that if you are on the central path you actually minimize the Langrangian with this value of lambda. This value here. Okay? So that's the picture.

Okay, that's great. If you minimize the Langrangian, then it means you actually have a point that's dual feasible. That's what it means. And that means immediately you're going to get a lower bound on your original optimization problem. So let's see what happens. It's actually kind of interesting. It says if you minimize this, which, by the way, you can do very easily for any value of t. This is Newton's methods to minimize this. But if you minimize this, then actually whether you like it or not, you will actually get a dual feasible point for the original problem. That's what's going to happen.

So you're going to minimize that, you're going to get. When you finish you're going to get this. You're going to define lambda i as minute 1/t, fi(x), like this. That will be lambda i. And you'll get a dual feasible point. And then of course you have a dual feasible point and you want to evaluate g. That's the dual function at that point and you work that out. So I'm going to plug this lambda into g.

Now g is nothing but this. Its f0. And I don't have it written here, but I'll write it out. Its going t be f0(x)+= lambda i, fi(x). Well, and then its plus nu transposed Ax- b, but that's going to be zero. So I'm just going to plug in the lambda i I found. This is going to be g of lambda. It's going to be that. But the lambda i, this thing is - 1/t, fi(x), okay? This f cancels that. That's a sum over the constraints, so this is equal to f0(x) +. Here that's all the same number, so it's actually – let's see, of the lambda i is that. So I get – actually, it looks like to me, - m/t period. Okay?

Now, that's very interesting. It says that if x minimizes this thing, that's something you can do by Newton's method. Then, I also inadvertently calculate dual feasible point, and that gave me the following lower bound.

And that tells me the following, that the optimal value of this problem is clearly as the following property, certainly it's less than f – I have a feasible point x, so the optimal value is clearly less than that. But it's bigger than g of lambda, but g of lambda is this, f0(x) – m/t. Well, it's beautiful. You know what that says? It says that if you minimize this function here – which is a completely smooth function. So that's Newton's method here. If you minimize this, the solution will be no more than m/t suboptimal for the original problem, okay?

So now, you have the hard proof that, in fact, as t goes to 8 you actually compute an optimal point. In fact, it's much more. The truth is that as you solve this problem and vary t you can say much more. You can say the following. When you solve this problem, you will produce not just a primal feasible point, you will produce a dual feasible point. And you'll really produce a primal dual feasible pair. And the gap associated with that pair, the duality gap, the difference between the primal value and the dual value will be exactly m/t. So that's what this means. You have a question?

Student: Should lambda be the minimum of the right insight instead of f0(x)?

Instructor (Stephen Boyd):What's that? You had the right hand? Should the?

Student: Should the lambda be the minimum of right insight over x, so that you?

Instructor (Stephen Boyd):Right, sorry, but the reason x, my x minimizes the Langrangian. You know how I know that? I know that because this is the optimality condition for minimizing the Langrangian for this choice of lambdas. So I do know that it minimizes. This is the, if I plug in my x*, it does give the emphemom over x of the Langrangian. I don't know if that made sense - it was a.

Student: Where did m come from?

Instructor (Stephen Boyd):M is the number of constraints. Right, so this is very – this is pretty cool. And by the way, there's a method for solving the original. We already have our first method, by the way, for using Newton's method to solve a constrained problem, and in fact, it's got a name. We'll see why it's called that later. It's called – so the name of this method is called Bount. We'll see later, because that's a takeoff on a name that we're going to see. Sount, which is sequential, so I'll tell you what this is. This is sequential, unconstrained minimization technique.

Oh, and I like the name. I'll tell you why. It's kind of a retro name, because it points out that everything we're doing so far was actually known in the 60s. And so when you mention this name it's sort of a reminder because you can type this into Google. Well actually, if you type this into Google you'll probably get a lecture I wrote, but below that you'll find whole books on this in the 60s.

That irritates the people who imaged that all of this was done in the last 15 years so – and that's why I like to use the term. So this is the unconstrained minimization technique, and it works like this. This is really dumb. Watch this. You want to solve this constrained problem. Minimize f0(x), subject to fi(x) < 0 and x = b. The Bount method says this. Pick t and solve the following problem. I'm assuming here – by the way, you have a strictly feasible starting point. We'll talk about how to get one if you don't. It says minimize this. F0(x) + 1/t + times minus sum minus, whoops, log minus fi, okay? That is a smooth function. Newton's method will minimize it – well I should say certainly in theory. I mean, so will the gradient method by the way, in theory, minimize it.

But the point is you can minimize this. When you minimize this you will have a point that is feasible here, and actually it will be strictly feasible. And you will – you will absolutely guarantee that you are no more than m/t suboptimal. So you might say done. That's it. We're just done. Pick your tolerance. You want one e minus six tolerance. Pick t, whatever it is. Ten to the sixth times m or something like that. End of story. In theory that's actually correct, so Newton's method just one shot does it.

We'll actually see. It's more of when you combine this with the homotopy idea that you actually start getting methods that are a, both practical and b, pass muster with our complexity theory friends, okay? So that describes the s.

By the way, you shouldn't make fun of the Bount, because there's plenty of practical applications where it's entirely appropriate just to pick a value of t, minimize that, end of story.

Student: On the previous yellow sheet where you rote the dual function, I was wondering how you got Ax – b(0)?

Instructor (Stephen Boyd):Well because, look. Ax = b is part of the optimality condition.

Student: Right, but that is going to zero, right?

Instructor (Stephen Boyd):No, it's not. No, it's zero. No, it's zero. Yes, I know. I solved here. I minimized. Here it is, there we go. I solved that problem, and the constraint is Ax = b, so I assure you that Ax = b. Ax = b is not negotiable here. We have a point that satisfies Ax = b, so. Okay.

Everybody got this? So this is – it's also pretty simple. We're up to about 1967 here. Yes?

Student: Ax = b there also?

Instructor (Stephen Boyd):To where? Oh, over here?

Student: Yeah.

Instructor (Stephen Boyd):Well - I should yes. You mean if I want it to be true?

Student: Yeah.

Instructor (Stephen Boyd):Yeah, sure. Whatever, okay. Sure there. Sure. These are details. You know you don't have to do that in a 300 level class, right? You can look, it's an official rule. I mean, I can switch a minus sign. There's some number above which you're not supposed to have, but it's expected. In fact, you wouldn't even want everything to be right here, would you? It would be insulting. This is a 300 level class where, you know?

So – but thanks for pointing it out. Okay, all right. So let's actually interpret this stuff via KKT condition because you can, right? You can actually interpret it lots of other ways. Here's another one. And in fact, this one is also like quite cool. It's this. Let's take – this is the point on the central path. This is xr(t) minimizes wherever it is. I've lost it. Here it is. It solves this problem here. Lambda * is the lambda defined by that. Let me point one thing out. When you fix t, you don't know lambda * until you calculated x(t). So you don't know what dual point you're going to end up with until you do this.

And curiously, though, you do know exactly what duality gap you're going to have. You're duality cap, your gap is going to be exactly m/t period. So that's – in fact, one way to say it is when you solve this problem – which appears to be a primal problem whether you like it or not, you're actually going to calculate a primal, dual pair for the original inequality constrained problem with a duality gap exactly equal to m/t. That's what you're going to do.

In fact, you can even think of t as a parameter, just, well, m/t is clearly simply the duality gap. So you can literally dial in whatever duality gap you want be done.

Okay, and that would be the Bount method. Okay. So let's look again at what it means to be central. Oh, I should say if you solve this problem you would refer to a point like that as central, or on the central path, or centered would be another one. These are the names you would use for that.

Okay. Well, here's what these points satisfy. Well, in fact, this one I can make stronger. You're actually strictly feasible obviously because, in fact, the objective that defines x *(t) has log minus fi. So this is strictly par.

And in fact, these are strictly positive. Okay, because those are 1/t time s - fi, which is strictly, so these are positive. Okay?

Now here’s the cool part. The optimality condition. Well, this is our definition of what lambda i is. We actually get - lambda i fi(x). This is if you’re on the central path is equal to 1/ t. Now what’s cool is the following. So we write down these conditions, and the optimality conditions for the original problem inequality constrained are really, really simple. And I will draw them right now – ready? It’s just that.

Now that’s true complimentary slackness is this, right? It says that lambda i, fi is zero. So if, in fact, you have x and lambda that satisfy. Well, now I do have to change this back like that, okay? These one through four with this zero here. That is the exact KKT conditions for the original inequality constrain problem. Okay?

If you’re on the central path you satisfy all four of these, except three is modified form exact complimentary slackness to approximate, and this is just 1/t. So that’s another way to say it. To say that when you have a point on the central path, you’ve actually found a set of points that satisfy three of the four KKT conditions and the one where they come up short is complimentary slackness. And in that one, instead on lambda i, fi being like minus lambda i, fi being small. I mean, zero, sorry, they’re all equal to 1/t. They’re all equal and they’re just 1/t.

So that’s another interpretation. This is actually the most common interpretation these days of these methods. So if you look at books and things on these methods they’ll just start from KKT.

Okay, now you can also think of this in terms of there’s a nice force field interpretation – and that’s this. Forget the equality constraints to make it easy. What we do is we have a force field here – well, we have a potential. So if you think of this as a potential, and I’m going to think of each of these with the minus sign as the potential contributed by each constraint. So each constraint right now has a barrier function, but I’ll think of that as a potential, which means a force field and the gradient of the force field is going to be - I’m sorry, the potential is obviously the force. So that’s the way I think of it.

Now, what happens is this. The forces balance at – I mean, when you minimize the potential the forces balance. And that means here that the force contributed by our objective here. So this is a force contributed by this is the potential. This is the thing you want to minimize, is actually balanced by the forces of the contributed by the barriers.

Now, the barrier forces have actually a beautiful interpretation. I mean, especially when you have linear constraints. That’s the easiest thing, so I’m going to do that. So let me describe how it works there. So here, for linear constraints – all right, that. Here’s the way it works. My force field, my force field on top of this is simply in the direction c.

And by the way, you can make this gravity. So this can be g times whatever down, okay? So that’s what this is. So it’s simply a constant force field in the direction minus c. That’s what this force field is, that’s the objective. So here’s our point were c – c is apparently this direction, so the force contributed by the objective is simply a constant force in this direction. It's pushing you towards values of low c transposed s, which is obviously what you want to do.

Now, the force field associated with the each log barrier term. You take the derivative of the log and you get one over something, and it turns out it's exactly this. It's an inverse distance, and so 1/r force field.

So this gives you a beautiful way to actually picture this, understand exactly how this works. Here's what you do. Take each constraint and spray it with some repellant, okay? And the repellant has the following property. Wherever you are here, this plane will put a force on you. It will point away from the plane. That's the direction it will point in, and its magnitude will be actually exactly one over the distance to that plane. Everybody got that?

So by the way, there are no such repellants or whatever. I tried thinking of like some case and some number of dimensions where you could spray positive charge. No – there are none. But anyway, imagine there were such a thing.

So you spray the – and this means, for example, if you're here. Let's actually work out what the force field is there. You feel a very strong force from this plane, and you feel a very strong one from this one. You're about equidistant – oh by the way, what kind of force do you feel from these three? Much less. So basically the force that you feel at this point is – in fact, someone tell me which way to draw it. Like, this way? This way? This way? Well, it's like that, right? So basically it would be forced like that.

And by the way – that's actually good. That's where the barrier force is not fighting the objective. It's actually pointing you away, because basically you're at the completely the wrong point, okay?

So and you can actually imagine all sorts of things. Like what if you get right up to one of these constraints? Which way is the force pointing? Well, it's basically pointing you away from the constraint, okay? All right.

Now, imagine – now, put no force on it. So in fact, what we'll do is we'll just make this in a plane. So this is actually this sitting level, and what happens now is the particle will settle to the analytic center. That's where all these forces balance. It will be like – you know, it will be a point there or there. I don't know. Somewhere in the center, right?

Now actually all I have to do is this. That's c. Let me see if I can get it right. Yeah – okay, so here's what you're going to do. Take this page and start tilting it, because that's exactly what you're doing, right? Because you're putting more and more gravity force on it, and when you tilt it all the way, you have to be able to tilt it like, I guess. Gravity has to be way, way high here.

But as you rotate this thing, what happens is there is a gravitational pull, pulling this in this direction minus c. And it will move over a bit for each angle it will hang off down a little bit until the repulsive forces from these three counterbalance the gravity force. Everybody see this?

Now as I crank the gravity force up super high you're going to settle over to a point here. You will – of course, you won't be touching these, but you'll be very, very near, and you'll be pinned right up against the active constraints. Near, but not actually touching the active constraints. And there, there repulsive forces will balance the attractive force generated by the objective, so.

This, it goes without saying, none of this actually matters. This is just for you to have a feeling for how this works, okay? Actually, let me just ask you a quick question to make you do get it. Let me ask you, suppose there was, how many constraints I did here. One, two, three, four, five. All right, you know what? I'm going to put some more constraints, ready? Like this, ready? Tons of constraints like that. Like, let's say ten of them. Okay?

Now let me point something out. The constraints I just added, they do not change the feasible set. So what's the optimal point? So of course, it's the same. It's right here. What does the central path look like now? Does it move? We haven't changed the feasible set. Has the central path moved? Where is it? Okay, what happens is these constraints, though they have no effect, they do not in any way change the feasible set. In any way. They are felt very strongly by this particle, because 1/r is a very slowly decaying force field, and what happens is the central path, the central point is probably. And this thing probably sneaks up on the optimal point that way.

Why? Because it's feeling a repulsive force from all these things, even though they're outside the feasible set. Everybody see this? I mean, this doesn't matter, but it's sort of just to give you a feeling for this.

Student: So as you tilt the plane, what is telling you to tilt it in the direction that actually follows the part?

Instructor (Stephen Boyd):I want to put a gravitational force in the friction minus. I'm going to put a force in the direction minus c.

Student: So c is going to determine how that?

Instructor (Stephen Boyd):C will tell you how to tilt the table, yeah. So you tilt it along c. Okay? So that's. Okay.

Now we're ready to tell you about the barrier methods – so the barrier method is this. By the way, the other name for this – there's lots of names for this. Sount, I already told you one. Definitely you should try to use this, because it's irritating to people who work on these things. And that's always good, so you should try to use this term. And you should also drop things like saying, oh wasn't all that known in 1969? And I'll tell you a story about this shortly.

Okay, so all right. So here it is. It's the barrier method, otherwise known as the Sount method. It's got other names too. It works like this – ready? Now, you start with a strictly feasible x. We'll talk about how to get one later. And some additional value of t, which is positive and some parameter mu, and some tolerance. And here's how it works.

You simply do centering steps so you can compute x*(t) by minimizing this. And this could be using Newton's method. I mean, it doesn't really matter, but if you want a complexity method, and if you want this method to work, well, you're going to have to use something like Newton's method here, okay?

So you do this and you will start from the previous value. The previous x* found. Then you update x. Now, when you've x r(t). Of course you're duality gap is exactly m/t. If m/t is less than your tolerance you quit – and by the way, you would return. When you quit, you could actually return actually both x *(t) and lambda * of t. In effect, it's considered polite if you write a solver in convex optimization – at least, it's considered polite to not just return a nearly primal optimal point, but to also return a dual, nearly dual optimal point with a duality gap that's less than some number, right?

Because that way, in fact, it's anyone can check. They don't have to trust you because – in fact if you've asked for a tolerance of epsilon, they return a primal dual point. None of your business how they calculated x lambda nu. Totally irrelevant. You just check the duality gap. And so they're actually returning essentially the suboptimal point and the certificate proving its epsilon suboptimal, okay? So you could do that here.

And then you increase t. And you would multiply t by mu. Now, let me just say a little bit about this. So the stopping criterion of course is completely noneuristic here, right? Because, in fact, it returns not just a suboptimal point with some vague idea that it might be optimal if some unknown parameter was small enough for something like that, which is how most of these method works. Instead, it returns not just the a suboptimal point, but a certificate proving certifying its epsilon sub-optimality.

And here you should imagine a tradeoff in the choice of mu. So the tradeoff would be something like this. Mu is how much you increase that parameter t in each step. You should imagine that things like 1.05. If you have 10.05, here's what should happen. Let's imagine you use Newton's method. Because important to get the tuition money. If you use Newton's method, if mu was 1.05 it should take maybe only two Newton steps to recalculate the new center, because basically you're at some point – well, let me draw a picture here. Basically here's what happens in – here's your central path. You're right here. You've calculated that point. You crank up t by 5 percent, and you want to calculate this, but you've just minimized the problem that was very, very similar. The truth is, where you're starting is nearly optimal anyway. And so basically in two steps of Newton's method you should be right back here, okay?

Now the problem is you only made 5 percent reduction in duality gap, because the duality gap is exactly m/t. If t went up by 1.05, your duality gap went down by 1.05. Of course, that means if you do it, I don't know, a hundred times it will work pretty well, or something like that.

So these are little mousy steps, and this is what you'd imagine. This is what a normal homotopy method is. A normal homotopy method you should imagine little mouse steps, because homotopy method says I have a parameter, I can solve this problem. I can't solve that problem, and what you do is you crank your parameter just a little tiny bit and you make each step. Not too hard. And you hope, you pray to the god of Newton, whoever controls Newton, whichever god's controlled Newton's method – things like that. [Inaudible] conversion, a couple of steps. That's a standard homotopy method.

However, it turns out here. It's going to turn out that the actual updates are very aggressive that can be used – oh by the way, this model of little mouse steps is exactly what the complexity theorists will tell you to do, so if you want. Later we're actually going to bound the total number of Newton's steps required and, in fact, the complexity theory will require us to take mouse steps there on the order of something like, you know, mu is going to be something like one plus and then one over d square root of the problem size, okay? That's what our complexity. And that choice of mu will optimize the overall bound on the number of Newton steps, okay?

In practice it turns out the values of mu can be very aggressive. They can be things like two, five, ten, fifty, depending on the problem. And these things will work very, very well. Shocking. That's really no homotopy step. These are just huge aggressive steps. So if you crank it up by ten, it means that you're going from this point. You're next point is going to be here. And you're next point will be even closer

So what happens then it might take you a bunch of Newton steps, so you're going to go way – you're going to have a whole bunch of Newton fun and you're going to land up there. Okay? So that's – this is not, you know, a normal homotopy, right, where normal homotopy is you just.

Okay, where I'll give you an example of a homotopy. How many people here use Spice? Well, so that's great. That just, for the record that was like no one here. Anyway, no it's a couple people.

Okay, to those people, so you know in Spice one of the things you have to do, you have a non-linear circuit and you want to calculate the bias, the equilibrium point – the bias thing. We have a set of horrible non-linear equations. I mean nobody, you know – you can't solve it. And you know already you can already have big trouble if you have, like, i-stable things like if it's got memory or stuff like that. Its multiple equilibrium points. But you want to calculate an equilibrium point. Here's the way you do it. So addressed to those people. Let's take an amplifier or something like that, and you want to know what the homotopy parameter is?

This is fantastic. It's totally cool. Anyone guess? Actually, I know a bunch – a lot of you people have been tortured with it, because a bunch of you people from EE. Now don't hide, don't think you can hide. I know a lot of you are in EE. You're in stat, you're safe. That group over there, but the rest of you there's no way. You know you've seen this stuff.

All right, so the homotopy parameter is the supply voltage. Because take an amplifier and let's hit the supply voltage to zero. Anyone want to tell me what the voltage and currents in the circuit are? You know, thank you. That was easy. Okay.

And then I take the supply voltage, I'm ramping it up to – what's your favorite supply voltage?

Student: Fifteen volts.

Instructor (Stephen Boyd):Fifteen volts – see that? No, no, no, no. I know what he's talking about. That dates him. I mean, you're way too young for that. That's crazy. That's like I might say that, plus or minus 15 volts. I can say that, but why can you say that? You can't say that?

Student:[Inaudible] Smith.

Instructor (Stephen Boyd):Oh, Cendren Smith, because you read old stuff. Okay, fine. All right, no. Your choices are five – depends on your age, okay? Or I guess the age of people whose stuff you read. Okay, so it could be five, 3.3, two, down. Now, if you're really cool you could say like 1.1 or .8 if you're super cool now, right? See, I know people knew and just holding out on. All right. It doesn't matter. So what happens is this. We take a big circuit. We set the supply voltage to zero. Everyone agrees.

All voltages and currents are zero, right? Okay, now you set the supply voltage to 100 millivolts. All right. What happens if you take like a big, horrible digital logic circuit and sent the supply volts to 100 millivolts? By the way, how ell does it work as a logic circuit? It doesn't work at all. Okay? However, can I calculate the – can I calculate the bias point? You know this. You know this, right? Come on, you held back on me on the satellite stuff, even through you're from Harben and should know this, right?

You've had this circuit stuff? A little bit, okay, all right. So take a logic circuit. DDD's 100 millivolts. What does the circuit look like? Oh very, very sub-threshold, right? It looks like a resistor circuit. So we can solve it in one Newton step or two, right?

No problem. See, you get thee things because the transistors, they're not even on. I mean, it's ridiculous. They're not even transistors yet. They're resistors, okay?

Then you go to 200 millivolts, but you start from the one you just found and then you go to 300, 400, whatever and so on. And if you go – by the way, from one volt to 1.3 volts and it fails to converge, then you go, oh. I'll try 1.15. Everybody got the idea? Anyway, so that's a – there you go. That's a homotopy method. That's how Spice works, okay?

Student: Is there any downside to using a large mu, like an aggressive [inaudible] or something?

Instructor (Stephen Boyd):Here? Student;

Yeah.

Instructor (Stephen Boyd):Yeah. I'll tell you what the tradeoff is. In a circuit problem you're trying to find the equilibrium. The downside is real simple. If you too aggressively update VDD, Newton's method, which is what is used to calculate the equilibrium position will fail. It'll fail. It'll just fail to converge, okay? So that's the tradeoff there. For us, we cannot fail because we solve a convex problem with every step. We cannot fail.

You can crank mu up, t up to the ten to the nine, and at least the theory says we can't fail because we can minimize any smooth convex function with equality constraints, right? So we can't fail, but what can happen if you update t super aggressively. Remember our theory, we can tot out all our theorems and talk about numbers of steps and blah, blah, blah. What'll happen is the number of steps will be just absolutely huge. Actually both in theory and in practice. So that's the problem.

Student: We can [inaudible], so.

Instructor (Stephen Boyd):We as a – yeah, Mantle doesn't do anything. Let's remember that. Okay. They do nothing. Remember, LA Pack is doing the work. Don't forget that, okay. Yes.

Student: The fact that the Newton method rolls over at some point in the transition?

Instructor (Stephen Boyd): Yeah, exactly.

Student: Do we use that to choose mu?

Instructor (Stephen Boyd):That's very interesting. There are methods. There's whole methods that actually do exactly this barrier method. And they choose mu so small that the can prove that when you update t by that amount you're still in the region of quadratic convergence. So that's called a short step method. There are lots of them, and that's a lot of the proofs go that way. And basically it says you can prove that by cranking up t by the small amount, if you were in the region of quadratic, if you were in the region of quadratic convergence before, you will remain in it for the new value of t and so on. So that's one way.

No, I can tell you how we pick the update. Well, I can tell you the reasonable way to pick it. Actually how you will pick it. Ready? Mu equals two. There. Here, I'll pick another one. Mu equals ten. How about mu equals 30. These are sort of the numbers that work. There are much more complicated ways if you get into ultra high-end ones. By the way, they don’t do a whole lot better, honestly, than these simple things. That’s the embarrassing part. So this whole list – you can get 50 papers written on how to update this parameter, right?

I mean, some of these are quite fancy and effective, at least for some problems. So this empirically – this appears to be the right, you know, anywhere between two and 50 seems to be the right number.

Okay. And there’s lots of eristics for the t(0). By the way, this is something that’s discussed in the book. You should actually read about that. T(0) actually implicitly is the original duality, your original duality gap. So that’s what t(0). M/t(0) is your original duality gap. So if you had some reason to believe that your current point was suboptimal by five, ten, it says a reasonable value of t(0) is on the order of one over five m or something like that. That that would be a reasonable one. That would give you the same duality gap and you’d just center. Okay.

So let’s do the convergence analysis. To show that it works is actually kind of silly. I mean, we know that, but we can say a lot of things. Now, the number of outer steps. By the way, the outer steps here are called – these are called centering steps, or outer iterations, or I don’t know. Whatever it is, right?

So by the way, it should be clear that you don’t need that this is, there’s lots of stuff that you could save here. For example, when you do a centering step, you don’t need to, like, polish off. You don’t have to compute x*(t) to 16 significant figures. That’s totally clear because, in fact, this whole thing is just a method to guide you toward a solution, right? So you don’t have to do full centering.

But, I mean, with Newton’s method it’s cheap. It’s a couple of Newton’s steps, and the difference between getting a okay solution and an unbelievably accurate solution, an amazingly accurate solution is two or three steps in Newton’s methods so you know, there’s no reason to worry about it too much, but okay.

So there’s a lot of stuff here you can imagine is not, can be shaped up. But for an algorithm that’s very short to write it’s amazing how well it works, okay. So how does this work? Well, the number of steps is – I mean, you can say the exact number. It’s very simple, it’s just log of – it’s log of m over epsilon zero. Every time the duality gap, after ht first step will be exactly m/t(0).

Then each time you do an outer step that shrinks by a factor of mu. So you can immediately just work out the exact number. It’s this. That’s the number of steps it’s going to take, and that’s ceiling of these things.

Okay? So that’s the number of outer steps period. Notice this has nothing to do with anything but t(0) and epsilon and m. Nothing else, okay. Now, the centering problem. Oh, and let’s look at how this varies with mu. If mu is big, this number is kind of small. If mu is like a little mouse step, like 1.01, this number is big. Right? I mean, mu is 1.01, log mu is .01, so basically that's 100 in the numerator here. Right? That's basically what this is. So what's that?

Student: Mu is [inaudible].

Instructor (Stephen Boyd):No. Log 1.01 is .01, kind of. Very close, right, so. Am I wrong? No. Okay, all right, so okay. Okay, that correctly reflects the variation in the number of outer steps is monotone decreasing in mu. So mu is your update aggression parameter, aggressiveness parameter. And that says that the more aggressive you are, the more duality gap reduction you get per step, so you take fewer outer steps.

Now the problem is, the more aggressive you are in your update, the harder the problems you're solving by Newton's method. So that's now – you can do this intuitively and just say therefore what we should do for our problem class is just try this out, adjust various, you know, solve giant families of problems.

Oh, by the way, Newton's method cannot fail. So we have a proof that Newton's method will converge and so on and so forth. And if you use the classical analysis, it would involve all sorts of constants that you don't know. But it doesn't matter. You would at least have, at that point, a conceptual proof of convergence, right? Which would be to say this method can't fail for something.

So that's fine. But, if you want to get – I mean, the modern results come by using self-concordance. So if f(0) and the log barrier are self-concordant, then tf(0) is self-concordant for t bigger than one. And tf(0) plus phi is self-concordant for t bigger than one.

And we'll get to that a bit later, but let's just look at some examples and see how this works. This is the dual of the problem you're solving on your homework, but it's an LP, with, you know, 100 inequalities and 50 variable. And this shows you the number of Newton steps. By the way, the computational effort is simply measured. The unit is the number of Newton steps period, so that's the cost is Newton steps here.

So the number of Newton steps here is plotted here, and this gives you your duality gap. Now – and so it's plotted with one of these staircase things, and the staircase thing. It says that the width of a stair is exactly the number of Newton steps required to carry out that centering.

The height of a stair is exactly the reduction in duality gap obtained with you finish re-centering. Now, in this method, the duality gap simply goes down by a factor mu, so since that's a log scale, the height of the steps for each of these plots is identical. And in fact it's exactly equal to log mu or something like that, or whatever it is. It's something. It's a factor of mu down.

So here's a little mouse steps here. They're not even that mousy. If you're doubling, but you can see everything. You can guess all sorts of things here. You asked about the

Newton quadratic convergence and stuff like that. You can sort of see down here. You're taking like one or two, probably one or two steps. I don't. Maybe these are two steps each, right? So basically you double mu, you down two Newton's steps, double mu, double t, sorry. Two Newton steps, double t. Do two Newton steps. That's this. Now, you're making pretty good progress down here, and you can see down here is what you get.

The weird part is here. This is mu equals 50. For mu equals 50, you see what's happening is the width of the treads are much wider, because you're taking more Newton steps to actually re-center.

But, of course, then the measure of progress actually is duality gap reduction per Newton step. That's really what you want, because when you start with a duality gap of 100 and you want to reduce your duality gap, which is your ignorance by a factor ten to the eight, then somehow you have to make progress. You have to reduce your ignorance by a factor of ten to the eight, and you have to do that in some kind of Newton steps or whatever.

So and you can see here a mu was 50. It's actually down. By the way, that number is actually like 30 something steps. That actually probably takes a little. This is probably a good time to stop before we get into all the horrible details of mu and t and self-concordance and all that. It's actually time to just take a moment and realize what this says.

By the way, you see this picture? It looks the same if there is a million variables. It looks the same if it's maximum entropy problem. It looks the same if this is a problem from finance. Signal processing, image processing, machine learning. They all look like that. They look just like this. Essentially no difference, okay? And it's completely independent the size of the problem. It just looks like this, okay?

Now this is really quite ridiculous because if mu equals 50, it means that with a total number on the order of 35 Newton steps you've actually solved an LP to extremely high accuracy using Newton steps. Actually before – let's step back and think about what that means. First of all, it's quite ridiculous. The feasible set is a polyhedron defined by 100 inequalities and 50 variables. How many vertices does such a polyhedron have? How many?

Student: Two to the fifty.

Instructor (Stephen Boyd):Two the fifty. Okay, sure, yeah, sure. That's fine. The answer is – by that you meant a lot, then I'll accept your answer. And that's a good answer. Two to the fifty. Okay. The answer is there's a lot of vertices on such a polyhedron. This is a baby, baby, baby problem. There are a lot of vertices on that polyhedron, right? Okay.

One of them generically is our optimal solution, okay? So what is insane is this method has done the following. Thirty-five times you stopped in the interior of the polyhedron, asked for directions and were told to go in some direction. You then tried a step of one. If you didn't like that you stepped off to a half and then to a quarter.

So somehow after 35 times asking directions you went from somewhere either middle of this polyhedron in our fifty to a point, which is essentially optimal. Everybody see what I'm saying? So the whole thing is highly implausible. Thirty-five steps, that's smaller than the dimension of the problem if you think about it. I mean, that's 50, there's 50 orthodial directions you can go into and you ask for directions 35 times. Since this big figure looks the same when there's a million variables, then it gets just totally beyond anything you can possibly imagine.

And it looks the same, which is ridiculous. It means basically to solve a convex problem with a million variables like 30-odd times you stop and ask for directions. And after 30 such steps you have an extremely accurate solution. It's really quite ridiculous when you about it from that point of view, so.

Student: How would this compare to using [inaudible] method?

Instructor (Stephen Boyd):Simplex method would also be a very, very fast on this. So simplex method is – I mean, it's nothing we're going to talk about, but simplex method is when you actually go along the outside of the polyhedron and you take an edge greenly. That actually works also stunningly well, so.

Student: So for each [inaudible] mu of, or mu squared? Instruction:

Oh, the cost per Newton step? Cost per Newton step depends on the problem, right? So in this case what's the cost per Newton step for this one? You have 50 variables, 100 inequalities so you have to. Yeah, it's something like n cubed. But in fact what you'll actually find, believe it or not, is forming the Hessian. It's more expensive. So it would be 100 times 50 squared.

Yeah, so that's what it would be. So the cost of a Newton step is 100 times 50 squared. I don't want – does someone want to calculate that? I guess – is Jung here or someone? Does someone want to calculate what the cost is? How fast can you do? Let's figure out how long this actually would have gone down. Let's say written in c solving LA pack directly.

So it's like 50, it's 100 times 50 squared. Let's just say that's what the complexity is. Should we just round it up and make it like 100 cubed? Sure, let's make it 100 cubed. That's four times mu d. That's okay. Let's make it 100 cubed. We'll round up, so 100 cubed. How long does it tae to solve an equation with 100 linear equation, 100 variables. I keep coming back to this, and I'm going to keep coming back until people start answering the right way.

Student:[Inaudible] millisecond.

Instructor (Stephen Boyd):Thank you. Millisecond. Okay. Sub-millisecond, okay? So how long did it take to solve this LP? Thirty milliseconds, okay? So by the way, it's not a lot of people who appreciate this fact because, well, why not? I'll go off on a weird tangent and why not?

Okay, so basically, you know, linear program has been around since 1948 or whatever, you know, and it's usually associated. People imagine – I mean, if you close you eyes you picture like a Cray computer and some guys wearing, like, white lab coats – and you know, its machine that costs a million. It's an expensive thing, and you imagine you would use it for, I don't know, scheduling the airlines tomorrow, pricing a derivative, designing a nuclear weapon. You know, you think of these big iron – you know? You know what I'm talking about?

That's kind of what you think. That's how a lot people think if linear programming, especially like if you're an older professor or something like that, and that's when you learn about all this stuff, right? So that's your picture of what solving an LP is, right?

And it's cool. You know, that's how United schedules their airlines tomorrow. In fact, that's how they do it. I mean, roughly, but okay. And, in fact, yes, that's how you optimize portfolios and things like that.

But I think what happened since then is, well, let's say Moore's law computers, I don't know, ten to the eighth times faster or something, and there's also been algorhythmic improvements. So the idea that you can solve an LP of this size in milliseconds I think really hasn't hit people yet. People don't know it. People who do control don't know it, for example. They sure don't know it. They still think you do, like, you can carry out four floating-point operations like per control calculation. They don't know it.

And people do this stuff like over in our own department here, they don't think about doing things like this. So this is important for embedded applications. So all right, enough of that. We'll move on. That was just a big aside. Okay, yeah.

Student: So what caused the mu to 150 in case it takes longer. Was that just because of all the haphazard?

Instructor (Stephen Boyd):You got it. No, no, no, no. The tradeoff – it makes perfect sense. If mu is too small, what's happening is this. You're making not enough progress each centering step. The centering steps are easy, but you're not decreasing your gap fast enough.

When mu is too large, you've bitten off more than Newton can chew. And so what's happening now is when you finish the centering step you've made a lot of reduction in gap, but what's happened is you're taking a whole lot of Newton steps.

Now the cool part is that this parameter. These two trade off each other, and you get this huge flat thing like this. It's not common to be something like this here. This flat – but the point is, there's a big choice of – this is not a parameter that needs to be tuned, let's put it that way. It's usually set and then forgotten. Another question?

Student:[Inaudible] to changing it over the problems?

Instructor (Stephen Boyd):Oh yeah, absolutely. Yeah, I know, you keep coming back. No, the truth is I'm showing and you will, in fact, code before next Tuesday and amateur version of this. It will be a very small number of lines.

No, fancy ones don't update by fixed parameter mu. You're exactly right. They actually have a very complicated way that, by the way, for LPs and QPs worked amazingly well. It's called like a marrow trip predictor corrector thing, but the shock is that something this stupid works like really quite decently.

So yes, sophisticated methods will actually adjust, will crank up t by some adaptive scheme. So yes, that's correct. There was another question. Do you have one?

Student:I was just going to ask [inaudible].

Instructor (Stephen Boyd):About what?

Student: About a pick? Like a small processor.

Instructor (Stephen Boyd):On a pick. I don't know – but, I mean, it wouldn't. I think it would be pretty. I don't see why we couldn't do it. I mean all you have to do is multiply the entire thing is that you have to multiply a matrix which is 100 by 50 by its transpose.

Student: Right.

Instructor (Stephen Boyd):Or all you really have to do is a QR on it, that's all, so yeah. You could put this on a , you know, arm or a pick or I don't know. I guess. I don't know that people have done that. That would be cool. Someone could do that next quarter as a project.

But some interior point thing on an arm processor or something weird like that for no reason, just because it would be cool. I don't know, okay. All right. So here's a geometric program, a GP. So it looks the same. Same story. And it's just the same. And then here's sort of what happens if you have a family of standard LPs, and then what's happening here is you're changing the size. So this -ism, what is this? Oh hey, this is your homework problem. What do you know?

Oh that's interesting, because you know what? Since all the code – it means Google will find the solution to your homework, but it doesn't matter. We all know that anyway.

Anyway, isn't this what you're doing? I think so, so yeah – fine, so. If you did it right here's what will happen. You'll make something that looks like this, and then this is the most amazing thing that happens. It's this. Here's the number of, what is m? M is the – what is m? Oh, a is a, r, m, by two m. So there's two, this is the number of constraints, the number of variables is twice this number. Okay, so that's 20 variables, ten constraints. One hundred variables, two hundred constraints.

Did I say that right? Yeah, I reversed those. Anyway, you get the idea. That's 2,000 variables, 1,000 constraints, and here you generate a whole bunch of random problems and then solve them. And this is the number of steps required to reduce the duality gap probably by ten to the minus six or something, or reduce it by a factor of ten to the eight.

When you look at this, you see something amazing, and that is that it's basically it's not growing. It just doesn't grow so and I can tell you a little bit. I'll [inaudible] story about this. I guess when these methods were first looked at, we'll see very soon. I mean, next lecture well see that the theoretical computational complexity says that the total number of Newton steps is less than the square root of m, where that's the number of constraints.

Okay? And by the way, with a number out front, that's just like a number. It's not one of these mysterious, useless convergence proofs that has, like, Lipchitz constant and all sorts of crap that you don't know. It's just a number, so it's going to be some number in front time square root m. That's the number of steps it takes, period, for these things.

Now, if you're a complexity theorists that you'd celebrate this because you'd say well, there you go. You see, it grows like the square root the problem size. Each step takes, you know, in the simplest case, you know, n cubed or something. And so you'd have an order n to the 3.5 , and you'd have a polynomial time party or something. A big celebration or something, right? And walk away.

Now what happened with that? And by the way, that's where the theorists – that's where they are right now. The complexity theorists will tell you it's the number of iterations grows like the square root, but after people see this on and on, and over and over again, actually somebody finally, timidly said no, it clearly grows less than square root m. And somebody finally said, I don't know, the fourth root. And then somebody said the log and finally. And what's become completely clear is that it grows like this, o of one.

Okay, that's become as an empirical fact that would appear to be the case. And in fact, to be much more specific, it appears to be between 20 and 80 steps for all problems. So let that sink in for a second because it's quite ridiculous, and it's actually the essence – it's why these interior point methods. And we'll look into them, and we'll look into centering and duality gap and complexity and all that, but I don't want the main message to be lost in all of that.

The main message is this. They take between 20 and 80 steps, the problem in finance, machine learning – doesn't make any difference. The size doesn't make any difference. It's about between 20 and 80 steps, and I think I may have told you this earlier. Maybe on the first day I told you this, but now you know more about it. We had a talk last year by a guy named Jack Ganzio from Kenbrook, and he came and solved some giant finance problem with one billion variables, okay? It was dense, by the way. No, seriously. It was some big tree. It had some structure, but this was not a small problem, let me say that. It wasn't super sparse or anything like that. It had a billion variables.

So he solved it and we were like, how do you do that? He had some room where they had a blue jean – you know, they have the whole room full of like thousands of like processors and all this kind of stuff. And I said what algorithm do you use? And he goes, the same one everyone uses. And by the way, not quite this. He used an adaptive steps size, but it's not a whole lot different. It's basically this.

Basically what you're going to write. Its a billion variable problem, and I said well, how did you pick the parameter value? He said, just the same everyone uses. You know, it doesn't matter anyway.

And I said how many steps did it take? He said 21. So basically, that's a single point. Oh no, I think I mentioned this earlier. It took 21 iterations. Now, by the way, each iteration took like several hours and the lights in Edinburgh dimmed when each iteration was carried out.

Well, I mean, it's solving a billion variables with, you know, a dense system blah, blah, blah. But the point is, in some giant room full of computers, right, with a big, you know, 12 kilovolt three-phase thing going in for awhile – but the point was this thing, at least we have one point layout here at ten to the nine provided by our friend, and – oh, well, this is using our unsophisticated methods. The sophisticated methods are down here at the 20 level, although that hardly seems to mater, okay? So question?

Student: This technique, like he's running in a roomful of computers, did this scale in [inaudible].

Instructor (Stephen Boyd):No, his was not a fully dense. His had structure – in fact, it was an event tree. He was doing stochastic programming for some finance problem or something like that. Expanded the event tree out by like 15 steps or something. That'll give you a billion variables.

So the question is how parallel. Let's talk about if you profile one of these what – in fact, if you run an interior point method. In fact, the one you're writing right now. What actually will it be doing? I mean, yours will actually be spending a lot more time doing interpreted overhead than actually doing any actual computing. But if you scaled yours up big enough, you do 20 steps, what are you actually doing each step? What is it?

Student: Linear systems.

Instructor (Stephen Boyd):You're solving linear systems, you're solving a Newton. You're computing a Newton step. Line search and stuff like that. It doesn't cost anything so, in fact, the correct thing to say is if you want to solve a convex optimization problem you're going to solve 20 liner equations. All right, 80, whatever, you know. Something between 20 and 80 liner equations, period. That's all you're doing.

And you can use any method. I mean, we've just looked at the most simple thing, right? Where you use a Chilesky or something. I mean, if you see something, if there's some structure right under our nose like banded or something like that, of course you should exploit it, but then there's this whole world of solving like huge linear equations that we haven't even touched on, but exist and can be applied, so.

Okay anymore questions about that? That was just like a weird, I don't know, detour. So okay. Let's talk about feasibility in phase one methods. So in fact, you'll have to do this. So the feasibly problem is you want to find an x for which fi(x) is less than zero. Well, I tell you the truth, is you really want to find something like that. If you want to initialize it in a simple interior point method like the one we have. So this is what you want.

So there's lots of ways to do this. They're actually all kind of cool. The standard tricks, these go back to 1948 or something like that. I mean, a standard trick is to solve phase one by solving a similar problem. So here's what you do. You solve this problem. This is the most classic one. You have a new variable s, and you say fi(x) is less than s here. Ax = b, and you minimize s here in this problem. That's a convex problem, too, however, for this problem I can always find a strictly feasible point.

Well, sorry, provided I know any x that's in the domain of f and satisfies Ax = b. I choose whatever x I like, okay? Not feasible for the original problem. I mean, a lot of the fi's are way bigger than zero. And then I just find the max of the si's and I take s equal to that max plus one, and that's now strictly feasible. Strictly feasible point. I applied a barrier method or Sount to this thing, and then I can put in a logic that says this, if s ever gets negative, just quit because that's good enough. You've actually produced.

And then you would drop to phase two. Phase two would start from there and would actually then optimize the original objective, which would be f(0). F(0) is not in here. Everybody see this? That's the idea.

So this is the standard. Now, if you're minimizing this problems and you minimize it, and s* is positive, you actually now have a proof that the original problem is infeasible, and in fact, the dual, by taking the dual point found from s* here by solving this problem actually says certificate establishing that the original system of inequality is infeasible.

Okay? And there's one thing where the results are ambiguous, and that's when p* is zero, okay? So that's when you're right on the boundary. It says that the problem is, let's see, it means it's feasible. Well, it could be all sorts of sick things could happen. It means fum could be feasible, but not strictly feasible, and so on and so forth. Okay.

Now, by the way, let's talk about how you imagine this is going to work. So let's do that. If the problem is feasible this will work perfectly. S will keep going down. At one point, s will become negative and you terminate. You put logic in there to terminate, so you call your barrier method with a strain that says phase one, and it knows then stop, not when your duality gap is low, but stop whenever s is negative and quit right there.

However, suppose it's infeasible and let's se what happens if something is infeasible. These are fi(x) here, and let's say I start with a problem like this. I don't know why I'm drawing them up like that. I don't know what that was, sorry. Do it again, all right.

So here's zero, our critical point. Here's fi(x), here's a bunch of inequalities that are satisfied like that, and here's a bunch that are violated, okay? Now, this phase one method puts s over here. It puts a bound above these guys. That's s, and it starts pushing this thing to the left, right? Okay, yeah. I was checking if that was the left, yeah. That's pathetic, isn't it?

Oh well, all right, so you start pushing this to the left, and of course if you ever push to zero, then they're all on the left. You exit and you jump to phase two, okay. Now what do you suppose is meant from infeasible. You're going to push and push and push and you're going to stop because you can't find a pint that has all the fi's negative.

When you stop, what do you imagine is going to happen? Suppose when you stop, s* is one. What do you think this plot is going to look like?

Student:A bunch of functions [inaudible].

Instructor (Stephen Boyd):You got it. A whole bunch of constraints are going to be violated by one. Okay? So a lot of constraints are going to be violated, because when you start pushing points like this, right, they'll pile up and an bunch of them will pile up and you'll keep pushing and it'll get harder. And then you'll come up against a wall and you stop.

Oh, by the way, what's the force that you have to push with? The Lagrange multipliers, right? So some of the Lagrange multipliers. Mechanically that's exactly what it is. So you push and push and push and push then you hit the wall. You hit the wall and a whole bunch of things are violated, okay?

Well, you answered your question. You're problem is not feasible, but there are other very interesting variations on this, and I'll talk about one and then we'll move – we'll quit, actually. Let's see. So the sum of infeasibilities method works like this - I introduce a private. Oh by the way, you can think of s as sort of like a little, the interpretation of s here is, you're looking for a point where fi(x) is less than zero. That's what you're looking for. So you say well look, I'll give you a little extra slack, a little extra bonus, you know. You don't have to do zero, you can make it less than seven, and then seven gets pushed down, okay?

So here though, everyone gets their own private little slack thing. By the way, if we've seen this before. This is the beginning of the story on the support vector machine, okay? Actually, it's the beginning of the story on a lot of things.

So what you do is this, and everybody gets this own slack. If this slack is zero, it means that inequality is happy. It's satisfied, and you want to minimize the sum of these infeasibilities. Now, already you should have all the neuroconnections before you even do this or anything like that you should have a feeling for what's going to happen when this is infeasible.

What's going to happen in this case is this is an eristic for solving the problem. Please find me a point where that violates as few of the inequalities as possible. Okay? This is an eristic for it. This is how the support vector machine works. That's how all these things work. Okay?

When you solve this problem, here's what happens. When you solve this, and if you have 100 linear inequalities and 50 variables infeasible, and this is what happens when you. I just plotted backwards I guess, but these are supposed to be positive. When you push these up here, here's what happens. You get to the point and basically most inequalities are violated. So this is the basic phase one. We use some of the infeasibility, there's only 21 in the end that are violated. Is 21 the minimum number of constraints that can be violated? Who knows? No one knows, but if the eristic for violating just a few.

So this would have lots and lots of applications. If you're doing engineering design, for example, and you have a whole bunch of inequalities and it's infeasible of course the main message to tell somebody which is useful is sorry, but your specs can't be met. That's the most important thing.

But after that, someone says gee, thanks for the bad news, but can you tell me which ones I should relax, you know, which constraints do I have to go back to somebody, the specificer and actually beg and grovel for more from, right? That's what basically happens.

So in a normal phase one, you know, everything is violated. You don't really have an idea of which were the ones that were more violating or something like that. In this case you would actually, in this case you'd actually go back and zoom in on one step. I mean, you don't know this is the minimum sets of violations, but it gives you much, much more information, so okay. We will quit here.

[End of Audio]

Duration: 77 minutes

ConvexOptimizationI-Lecture19

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Student: Yes.

Instructor (Stephen Boyd):How much.

Student: It went up to 10.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Student: What is the m again?

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Student:30.

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

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

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

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

Student: It comes in in computing the Hessian.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Student:[Inaudible].

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

Student:[Inaudible].

Instructor (Stephen Boyd):What's that?

Student:[Inaudible].

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[End of Audio]

Duration: 75 minutes