ConvexOptimizationII-Lecture09

Instructor (Stephen Boyd):We should make sure the pile of projects is somewhere. Who's got them?

I guess we'll start. I guess the most obvious thing, I guess most of you have figured out by now is we have looked through these preliminary project proposals, and they're floating around so make sure you get yours. Do not throw those away because we didn't copy them and we wanna make sure there's progress, so when you resubmit them, we want to look at the old one again too; and then judge whether enough progress was made to justify our even looking at it again.

So let me make a couple of – the big announcements, and then I have a list of complaints. I just happened to have read all of them this morning. Again, this morning in order, so; but let me say a couple of things about it. They were okay; we marked up a lot of things. You might not be able to ready what we have so you have to be in touch with us on these, so they'll all be resubmitted.

A few were in really good shape and some of them we just sort of, we didn't even get it, so, or there's bad notations, so we just kind of marked it up and then stopped. So what we'll do is I think it's next week, nominally, is that correct? The end of next week it's the mid – what do we call it? The mid-term something or other, so by the end of next week, you'll resubmit. By the end of next week, it should not just be a rewrite. First of all, a lot of them just need serious editing. Some of those, by the way, you could get to us earlier. That would actually not be a bad thing. But by the end of next week, it should actually start having – I mean, there should be a problem. You should have tried some of these things. I mean, these days with things like CVX, very, very easy to actually do these things. It's faster to actually write code than it is to write a Lay-Tech and English description of it that's coherent. So by next week you should have done that. So the way that's gonna work, and we'll announce this on the website, is that you'll be turning in your – I guess the next round of these things next week. If we didn't like your typesetting or something like that, turn that in immediately to us because there's just no reason for us to read things that are sloppy, so okay.

So what I'm gonna do now, I know this is weird, but I'm just gonna rant for about five minutes, then I'll be fine and then everything will be okay and then we'll move on and stuff like that. So here are things. I'm gonna say them once and then I don't, I turn it back in, I don't want to ever see any of these again and anything that comes in from anybody in this class. It actually drives me nuts if I see it from anybody. All right, so I'm just gonna say it once just so it's said and official and on the record. I have a list. So the first thing is capitalization. We found this completely random capitalization. Sections, half capitalized, half-lower-case, unacceptable. So choose a consistent style and stick with it. We don't care what it is, there are many choices. Titles; title of a paper is generally all caps. By the way, you should use our template. We didn't put it up there to not be used. So I know some of you decided to typeset things your own way, don't. Typeset things our way. Your way, it's okay, there are other styles but in this class, you'll use our style,

period. It's just easier for us and for everybody involved. So the capitalization is silly. Look at section titles, I mean, and subsections, and there are; this is the standard, and so if you decide that your capitalization is this for a section, the firs word is capitalized and nothing else, fine. Make it that way for every section, okay. I mean, but it's totally unacceptable to have these, and you'll see us mark these things in these things.

The other thing is problem format, so a lot of people did use our templates. Other people decided to typeset things themselves, don't do it. Use our method, so I don't want to see Macs and then S-key and some weird thing where it was right justified or something. I don't want to see your creative Lay-Teching. It should be our format, period. So basically if your project, if the thing you produced doesn't look like the materials we produce for this class, we're not gonna read it. So just know we've marked these – I don't want to see your Lay-Tech things. For Lay-Tech, there are some really basic things I want to talk about. I mean, these are horrible crimes in Lay-Tech. One is to typeset a mathematical symbol. I mean, to use a mathematical symbol and to put it Roman. Like to say yeah, so the variable x, which is just the letter X. Come on, this is amateurish. Some of you did this, not many. Clean this up instantly. We don't want to see it. I mean, that's like so basic. Now the flipside, I'll explain a little bit. So the flipside is this; when you have a subscript or a superscript or something like that and if it represents or derives from English or any other spoken language, then it is not to be typeset in math, okay. So for example, if you want to have a – for some reason you want to talk about some f as some variable and you want to talk about the sub-optimal point, chosen by some scheme you've come up with, and you want it to look like this, let's say, okay. You want it to look like that. That is math, that is English or derives from English and therefore, the only correct way to do this in math mode is this. That's – and then this is math RN; so that's the only way to do it, okay.

So that's the only way to do it. So, and I'm gonna say these once and I don't ever want to see these again. I want to see these looking right. If you look through everything I write and other people who write well; if you look at that, you'll find these things are followed flawlessly. And by the way, if you want to see an example of a good, perfect style, look at Kenuth here, because that's a perfect, perfect – there are no exceptions of any kind in anything he's ever written. It's not exactly, it's not the format I use, but it is flawless. Just take a look at it. You will find zero inconsistencies. For example, he has a rule which I try to follow, which is this: no sentence starts with a mathematical symbol, for example. Let's see, a couple of other things. Okay, if we go to – for the typesetting just follow our format. It – full page, 12-point article. I don't want to see cube column and your fancy thing. I don't want to see a line and everything. I know how to make an H-rule too. We don't want to see it, so just use ours. It's generic and that's fine; and also we have a rule that when you declare a problem, you must say what the variables are and what the data are.

Also, I guess just some basic things. Oh, let me mention one other thing, sorry. Punctuation in equations, so this is something that's actually not done right in a lot of cases, but there's no reason you shouldn't do it. You cannot just have an equation out in the middle of nowhere. You can't finish a sentence and then all of a sudden write y = ax

+ b. You just can't do it, it makes no sense. It's not English, so the correct structure is something like this. If you say well, we have an output y and we have state x and blah, blah, blah and they are related by, then you can have an equation that says ax = b. If that's the end of the sentence, there's a period after that thing because the type of ax = b, the syntactic format is, I mean, in that case it is itself a little phrase and it has to have period after it. I mean, if you want to say they are related by, then equation ax = b, you have a comma, and you can say where a is the blah, blah, blah, blah, blah, that's fine. There has to be a comma there, so this is something – it's just, I mean, I don't see any reason why you shouldn't just write this way. There's just no reason not to.

Okay, oh, back to the – when you state a problem, No. 1 use our format, that will come by using, just following our template. So, you'll follow our template, but then you must say what are the variables. Oh, by the way, another of the most amateurish possible Lay-Tech error is to put a – I mean, this is just complete amateur stuff – is to put a blank line after an equation, which will, of course, in Lay-Tech, generate a paragraph, okay. So there's no reason for us to see things like that, right. There's just absolutely no reason for us to things like this. To see y = ax + b, something like this, blank line and then this, okay. There's just absolutely no reason. This is just like, this, is – in fact, here, let me make it worse. That's the most – we're not even gonna accept this, right, because this is – I mean, that's wrong, that needs to be like that. This blank line is completely wrong; this blank line has a meaning. It's interpreted by Lay-Tech; it's not what you want and all that kind of stuff. So things like this we don't want to see. Oh, the other thing is just like spelling errors. No reason for us to accept spelling errors, right. So Bib-Tech, use Bib-Tech and use it right. So, not only must your references be in Bib-Tech, but it must be done correctly, and I want to mention a few things here, because these are actually things that are reasonable. Let me say what they are. You must use Bib-Tech, don't trust if you go to Google Scholar or something like that or Site Seer and you import Bib-Tech.

A lot of those are amateurishly done and incorrectly done. Read the manual, I mean, read the first page of the Lay-Tech stuff. Read about Bib-Tech. There are mistakes when you get them off of like Google Scholar, and there are some conventions that depend on different, like the Commonwealth English speaking or whatever. So let me show you what some of those are because they're wrong. This is not right, so this is not right; nor is this with the periods, although in British or U.K. convention that's okay. It's not okay in this class. So that actually should be this, AR. The other thing, of course, but again you have to look at this, use this. The other thing is in Bib-Tech you need to know to escape things; otherwise they'll be made lower case. So Bib-Tech will ignore your – it's gonna ignore your capitalization, it'll enforce its own because it knows it can do it more consistently than you can, but it won't know to capitalize things that you don't tell it. So for example, if there's a colon or something in a title and you want the next thing to be capitalized, then this has to be escape, okay. Like that, if you wanted that to be, you're gonna have to escape it. So, okay; so these are – I'm just going over, for those of you who came in late, I'm just ranting because I read 30 projects in a row this morning to review them all. Okay, now, for some of these we never found out what the problem was. We got a lot of background, we heard about this, that and the other thing; we never found out what the problem was. So what we want is we want to get quickly to something that

says, here's the problem, and it should be a mathematical format. Say, you can have some stuff; we're looking for this, that and the other thing. The next – we want to see quickly what is the mathematical problem. The idea is that that problem should be understandable by anybody who is in a mathematically sophisticated field. That includes math, applied math, computational mathematics and engineering, anyone in electrical engineering, computer science, statistics. So you want to write high BBC style math, right, that's understandable. Nice accent, understandable by everyone across the world who does math. So it means you can't have – it's not like you're telling a story. Like you say well, we did this, then we added that and oh, and I forgot about this. Did we tell you about this? And also we want this to be smooth. Oh, and also there's these other constraints.

No, you're gonna get to the problem and say – so you can say you were doing portfolio optimization. You can have a paragraph that's English; I view this, I see it in formal XML. So I see an English description that says we want to do portfolio, you have a number of investments, notice this is English, a vary and mean return and the riskiness of the return, they're correlated and blah, blah, blah, and you say the problem is to allocate, to make an investment that whatever, maximizes return subject to some down [inaudible] risk. That's English, then it starts. Let x denote, that's if you following the Kenuth rule that a symbol does not start a sentence. You say, let xi denote the amount invested in asset i, but there should be a problem there that anyone in math, a random person in the math department you can grab and say, read this and they should look at it, read the first part and say, I don't know anything about investments; but they should read that second part, read your thing and know exactly what the problem is. Everyone see what I'm saying? So that's the high BBC style. It shouldn't have any jargon from some field or something like that. It's just readable by everyone. So okay, in a line we never really – in a bunch of the proposals, we never really got that crisp statement. That crisp statement that I can take to, for example, Brad Osgoode or one of my friends in statistics or anybody I want on campus who, or actually throughout the world, who is sort of, is trained in any mathematical field, and say do you understand it? They'd say, well look, I don't know anything about finance, but I sure understand – that problem I understand perfectly. There should be no undefined symbols and so on. Okay, the other thing is that we want –a lot of times we got the problem, but we didn't really get the approach. That's okay, because that was a preliminary proposal for your project, so but when you resubmit, we'd like to hear a little bit more, we want to hear about the approach because you should not be at this point – you should have thought a little bit about approaches.

Oh and then I'll get back to something. Oh, and the other thing I want to say, please never mix the problem statement method and approach, ever. So these must be separated by paragraphs, or even better like sections. In fact, a very good way to do it is to label each paragraph with, in fact, the paragraph label in Lay-Tech. This is actually a very good way to do it, in developing a document. That's typeset in bold, it's a paragraph title, it's like a sub, sub, sub-section, and you write what the paragraph is about. You can later come back and comment all these out as you generate the final product, but if you can't write something like this for each paragraph, it's not well written; and you could say something like the approach, our problem, background or other methods. So, I often start

by writing everything this way and then these get commented out in the end. No one knows they're there, but this is a very good way to develop stuff.

So I think this may finish my – that finishes my ranting and ravings. Actually, we think some of the projects are actually going to be – I mean, a bunch of them are gonna be really good, so it should be fun. Just come and talk to us. These minor things, like I said, like Bib-Tech, all that, putting it in our format, misspellings, you deal with that; and then if you want, bring the clean version back to us and then we'll talk. But do that quickly so we can give you some real content. So if we stopped reading because we didn't like your typesetting for example, which I think is a legitimate reason to stop reading, then come back and – fix it and then come back to us and we'll talk about the content and stuff like that. So okay, so that's it. Okay, so that finishes my rant. Now, actually some serious stuff; we finished last night two lectures that may be useful to some people. They're preliminary, we haven't posted the code yet, but we will. One is on sequential convex optimization, so; and we'll post the code as soon as it's clean enough. So this is for people doing non-convex problems. I know there's a couple, Mike for example, or you guys, so read those, and honestly for you guys it would have been better if we'd gotten that lecture done earlier, but its fine. But go and look at that, it should give you some rough idea about the ideas of sequential convex optimization and we have another one that's on MPC, model predictive control. So, and I don't know, I'm not sure that any projects are directly using that, but you might. You could, if you wanted to go the stochastic case, so, but it's related. So okay, okay, don't mind me. We'll continue.

Today is my big office-hours day, so I'll be around all day and please come by, and if you want to talk first, if you want to get me today for content style feedback, before you go and fix all of the Bib-Tech, Lay-Tech, all that kind of stuff, that's fine. Okay, let's continue with decomposition methods; so, okay. So decomposition methods, we looked before. It's sort of a generalization of this idea that if you had a problem that was completely separable like this, you would solve the few parts separately and the idea in a decomposition method is you should imagine a problem where the problem is almost separable, meaning that you can imagine x1 has dimension of 1,000, xq has a dimension of 1,000 and y has a dimension of 5.

And so the idea is you have a problem with 2,005 variables, but it's got a weird sparsity pattern. The sparsity pattern is that only five variables kind of couple across these, so you should see a graph with like maybe two full graphs and like five little edges or something in the middle or five nodes in the middle that kind of connect to both sides. So, and that's a canonical ones. None of the mathematics requires anything, one thing to be one size or another because it all works no matter what, but that's sort of the picture and the question is how to solve this in that case. And we looked at primal decomposition, primal decomposition is simple. You simply fix this coupling variable and you say, and you minimize the first one and the second one; but now they can be done separately because once you fix the y, like I like to think of it in terms of that graph. When you fix y, you are actually sort of removing those nodes as variables and then the graph disconnects and you've got two, let's say, dense graphs and those can then be solved separately once it disconnects. Now the problem is that if you fix y, and you minimize that, minimize that,

there's no reason to believe that's the global optimum, but if you optimize over y, you do get the global optimum.

So primal decomposition works by this method, you simply broadcast what y is as a complicating variable. You have a question?

Student:[Inaudible].

Instructor (Stephen Boyd):Yeah, right.

Student: It can be a very complicated variant, right?

Instructor (Stephen Boyd):Right, so that's a very good question. How do you get expressions for fi1 and fi2? So in some applications you really do get expressions for fi1 and fi2. In the vast majority of applications, there's no expression for fi1 and fi2. They're not – in fact, as we move forward, the interesting thing to find out is, how will be access fi1 and fi2? So we're not generally looking for formulas for them. The access to fi1 and fi2 is going to be the following.

We're gonna evaluate fi at a particular point, so it requires only that we evaluate fi and there's one other thing, there's one other method we need to implement, is we need to implement fi.subgradient, okay; so we have to get a subgradient of fi. So in this case, if – let's suppose that this represents something like a linear program or something like that.

So this is a linear program, then in fact this function is piece y, is like piece y is linear or something like that, but it's fantastically complicated, right. Even a linear program with like ten variables and 50 constraints, the optimal solution is indeed piece y is linear, or a quadratic program even. Quadratic program is piece y is linear, but the number of sort of regions it has, so if you want to write sort of an expression for it would be like 1030 or something, but you're not gonna write an expression for it.

What you need to do is, you need the ability to evaluate fi1 of y and a subgradient; so it's very important to think about. We write the symbol, but the question is what do you actually have to implement to carry out the methods. So, and the answer is you have to evaluate and evaluate a subgradient. Those are the two things we have to do.

Okay, so that's a distance, so you see, here's what actually happens. You actually have to minimize this function in calculating subgradient, so that's what you have to do. In particular, you do not have to form an expression. Now, there are cases where you form an expression. I mean, if for example, this is like quadratic problem – by the way if you have a quadratic, pure quadratic unconstrained problem, convex, and you minimize over one set of variables, what's the result in the remaining variable?

Student:[Inaudible].

Instructor (Stephen Boyd):They have a quadratic function. It's this problem exactly right, so this function is quadratic jointly in x1 and y, and then I'm asked to optimize over x1. You minimize a quadratic, which you can do by linear equation, and what is the remaining function? If you minimize over a quadratic, it's quadratic and what is the quadratic form? Now, for a quadratic you can give an expression, because you can give a matrix. What is the matrix? Sheer complement, exactly.

Okay, so that would be a case where there's an explicit form for how to do this. You minimize this thing and you actually form a sheer complement and stuff like that. That pretty much wraps it up. That's pretty much the only case where there's an explicit form.

Okay, this is final decomposition and we looked at an example and we got the dual decomposition and then this is where we'll start today. So, this is how dual decomposition works. It is very interesting. It turns out it's different, but I'll tell you what it is. Again we want to minimize f1 and x1y plus f2 of x2y and what we do is we now split y into two variables, but we insist that they're equal. Now, this is the kind of thing you do in duality all the time, where you make some incredibly stupid transformation and you think, how can a transformation this stupid actually lead to anything good? And anyway, we can see then it does.

So here's what we do. This is, by the way, if you like to think of it, I like to think of y1 as a local version of the complicating variable, y2 as a local version and then this is a consistency constraint. So, if for example, you are sharing area and power, for example, an area and power budget in designing an integrated circuit and there are two subsystems here, then y1 might be the amount, the number of milliwatts that this guy gets. Let's jus make it power, and then y2 over here, is actually also the number of milliwatts this guy gets. So this guy gets some power budget minus y2 or something like that.

So basically what they do is they would both have a version of what this is, and this consistent, this means consistent. Okay, now we do this, we form a Lagrangean and you get this first function plus that. That's the objective plus new transposed times the quality constraint and now what you realize is that L is separable. It splits into x1,y1 and x2,y2 because these separate precisely because we split y into two versions, here local versions and then this thing is linear and so it's separable.

So it splits, and what that says is that if you are asked to minimize L, you can actually split it and do it independently. At two processors, two locations, however you want to visualize why it's good to do it separate. These separate and so g1 you can calculate separately from g2, like this, and we can actually have all sorts of nice meetings of what new is here. We can think of new as a price, for example, here. New is a – we'll see later if we can think of new as a price.

Okay, now the dual problem then is to maximize this thing, and now I have two functions that can be evaluated separately and so in dual decomposition here's what happens. You distribute not a resource allocation to the two subsystems, but you publish your price; new, and then each one uses the price to calculate an action and then that's what it does.

This is actually an – this says if you like, if that's a cost – the news by the way can be either sign, so they can be prices or subsidies. Whatever you want, but what this basically says is, go ahead and use as much y1 as you like or as little. Please, go ahead, but you have to account for it in your accounting and for example, this new transposed y1 says these are the prices, and this says basically you're paying for it and then it becomes a subsidy for the other one. So that's how that works, and the mater, the dual problem manipulates the prices and you can see clearly what happens. If you crank a component of new up, that's gonna tell this guy, that component of y1 is more expensive and then when it resolves this problem, presumably it'll arrive at a point where there's less y1, less of that component of y1 being used. So that's the idea, okay. Now subgradient of either of these is actually nothing but the residual and that comes from stuff we've done before, but here's dual decomposition, and this is using subgradient method, but you could use ellipsoid method, analytic center cutting-plane, anything you want; any method for non-differential optimization. So it works like this, you solve these dual sub-problems, then you update the dual variables. This is the subgradient update and what happens in this case, is at each step you get a valid lower bound on the optimal price, but you don't have feasibility. You only appreciate feasibility in the limit in dual decomposition; so, which is interesting. Actually, if you're ever feasible here, you stop, because then you're off. So, okay. So let me say a little bit about how you find feasible iterates. When you have –at each iteration you maintain an x1 and a y1 and an x2 and a y2, and it's not feasible in general, meaning that you don't have y1 as y2. So basically your two systems have sort of designed themselves separately. They have a variable that is supposed to be the same, but they're not the same and so the idea is to adjust prices to drive the inconsistency to zero, to drive the two to consistency. However, if they stop for a minute, I mean, if I have one electronic sale driving another, for example, this one might design itself assuming it's driving, for example, a capacitive load of whatever, 10 femtofarads or whatever.

This one might design itself with the requirement that it present a load that is 12 femtofarads, or whatever, so they're off. And if you put them together it won't work, because this guy's driving a higher capacitive load than he was designed for. See what I'm saying? So what you do is you just start, you adjust the prices to adjust these so that these 10 and 12 femtofarads, so the loads actually come into consistency. When they hit consistency, they're done, they're absolutely done. When this guy is driving a 10.3 femtofarad load, was designed for that and the other one has designed that sell to present a 10.3 femtofarad load, you're done. That's the picture, okay. Now, what you can do, is when you've got an iterate here, a very good guess of a feasible point is sort of the average, like this. You take y bar, and this is essentially the projection of this pair of infeasible points onto the feasible set, because the feasible set is just y1 = y2. That gives you an upper bound, but it might be this could be infinity in general cases, in which case it's still an upper bound, it's just not a really informative one. So, okay, now you can also do this. If you have a method that says f1, if you have a primal interface to f1 and f2, you can actually do things – like a prima interface says I'm fixing your interface variables, please optimize your internals to minimize your costs. So, if I make the call to f1's primal interface, if I say, please fix – primal interface means this is exactly the interface you'd use if you were running primal decomposition.

Primal decomposition says I'm assigning you this variable. Now you go and optimize your private variables to deal with it. Dual decomposition says; use whatever you like on these interface or public variables. I'm charging you for it for subsidizing before. That's dual decomposition, so if you are actually able to minimize this over x1 with y bar, that's a primal interface, then you can get a better upper bound here. You have to do better because this one kept x1 and x2 the same. Okay, so this is the same old example we did last time. I mean, it's a trivial example, but here it is. So this is now a function of the – this is the price, or the dual variable, and the first system as a g that looks like that, a dual function and the second system looks like this, or this is the first. It doesn't matter, and you add them up and you maximize and I guess the optimum is somewhere around in here like that. So that's the picture. Okay, and this would be dual decomposition using bisection, and let's look at dual decomposition at every step always gives you a lower bound. Looks like the optimum value is a little bit more than 1.7. This is that crappy lower bound where you simply run dual decomposition, average the two y's and see what the objective is and you can see, by the way, that by here you could terminate with a pretty – with this known duality gap.

If you actually have a primal interface to the subsystems, then you can see that actually you could stop after – I guess that's three steps. Essentially, after three steps, you have done two things. You have actually produced a feasible solution to an objective of whatever that is, 1.71, let's say; and you have produced a lower bound from duality that's like 1.705 or something like that. So for many problems that's called quitting time. So, that's how this works. Now, this one would require both a primal – this would require a primal interface to the two subsystems and this requires a dual interface, so this is how this would work. Okay, so the interpretation I think we've already gone over this, but here's one. You can think of y1 as a vector of resources consumed by the first unit and y2 is the one generated by the second and you have to have something like supply equals demand. Then you set new as a set of prices here and the master algorithm adjusts the prices at each step rather than allocating resources directly. That's what primal decomposition does, and the reason you get the minus and the plus is that one is a supplier and one is a consumer. So, you go back over here to look at what happens. Here you go. If you want to know why the sign difference here, like this, it's because one these is going to be interpreted as a supplier of the goods or the resources and the other as a consumer. So one gets revenue by selling it, and the other gets – pays to use it and that's how that works. So this is sort of the – and as you can imagine, this is all very basic in economics and stuff like that. It's not this clearly stated, by the way, in economics, but oops, I'm not supposed to say that. Sorry, it's just a different language. Okay, so that's that, okay.

Okay, now we're gonna look at what happens when we have constraints. And it's not much different here, so let's see how that works. Let's imagine that we have – now, the constraints we can group into two things. These are to be thought of as private constraints, local to subsystem one, and these are private constraints local to subsystem two. And then we – now these are the constraints that couple. Now by the way, in this problem here, these two problems are actually completely separable. They're only coupled through these constraints, because here, I mean, you could make a more

complicated, general form where they're coupled through objective and constraints, but just to make it simple, we're gonna couple only through constraints. So this one is literally the shared resource problem. So for example, that's subsystem one, or one unit of a company or something like that. That's subsystem two, another unit of a company, or that's like cell one in an integrated circuit, cell two or something like that or in some system and these are shared resources. This says, for example, the total power has to be less than 30 milliwatts and the total area has to be less than a millimeter square or your total warehouse space has to be this, or some regulatory thing that covers everything. So that's it, and then the real question then is to figure out how to allocate these shared resources between the two systems, right. So for example, the first component of this says that the sum of two numbers has to be less than zero, and the question is well, how do you do that? Do you make this one -1 and this one +1? Do you make this one +5 and this one -5? That kind of thing, that's the question of the resource allocation. Or do you make them just zero and zero, so that's the question.

Okay, all right, so this is the picture, and primal decomposition again, it's kind of obvious. It basically says the following. You introduce a variable t, which is a fixed resource allocation and you simply say, t is the number of resources you're gonna throw at h1. Sorry, at the first subsystem. So you say, okay, you get this much warehouse space, this much cash, this much this and this much power or whatever you like, it doesn't matter. These are your fixed resources and then this sub-problem one says with these resources fixed by the central office, you go and operate yourself as optimally, given these resource allocations. Well, if you give t to this guy, you have to sort of take it away from over here, because the sum are zero, each component. So here it means you put, you take it away over here. Well, take it away if the entry is positive. If it's negative here, you've taken it from this guy and you've given it to this one. So you say minimize this second system subject to this and the t and –t guarantees that if you add these two together, it's less than zero. That's how you've done; you have an explicit decomposition of the resources. You've said you get 30 milliwatts; you get 70, just period. Now it says deal with it. One possibility, of course, is that these are infeasible, because this thing needs more than 30 milliwatts, let's say. That's no problem, because this will simply return the value of fi1 to infinity. So that's simple enough, and the master problem is to minimize the optimal value of this thing plus the optimal of that one over t, and this is sort of the idea here and the cool part is that once you've fixed t, these can be done separately. Okay, so final decomposition works like this. You have a value of t, you solve the two sub-problems, and you have to find the Lagrange multiplier, Lambda. To find a Lagrange multiplier – sorry, a subgradient. I've already given it away. So here, if you wanted to find out a subgradient for this problem – it's absolutely classic. If you stare at this long enough you realize that t is a vector here.

This is actually something you study in 364a. It's the optimal – it gives you the Trainoff curve of optimal cause as a function of these resources. It's convex in t, that's clear; but the other thing is that if you calculate the Lagrange – if you solve this problem privately and then look at the Lagrange multiplier here, for this thing, if you call that Lambda one, that gives you exactly a subgradient of this function. Here you get a Lagrange multiplier and because that's a minus sign, it switches the sign. It's minus. Okay, so find Lagrange

multiplier and Lambda one and Lambda two, and I'll give an interpretation of what these are in a minute, and then it says you update the resource allocation, and so that basically says you update t. This is in pure subgradient, you could use anything else. You could use ellipsoid, any of these other things, smooth subgradient, all these things. This updates that. These are usually the easiest to describe, if not the one that works best, but that's it. So, okay, now this gives you a subgradient of the sum and in this case all the iterates are feasible. I mean, that's assuming that these are finite, so if you have a value of t, that works and this is, these are – yeah, when the sub-problems are feasible, it's finite.

Okay, so here we go. This is two problems. Let's see. There's two quadratic functions, the ci are polyhedra and they're sharing two resources here. There are two resources that they have to divide up and this shows the progress of the cost minus p*. So this is as the iteration goes, you can see that they're already – it's quite close to optimal in terms of sharing, and what's being worked out is how these two resources are split between these two subsystems. And here's the resource allocation. They just start both at zero and then immediately the subsystem one, that's the blue one, starts getting more of the resource. By the way, it's beautiful to ask what – I mean, there's a beautiful interpretation of this. Let's make this scalar, just make it just scalar. So t is a scalar, a single resource is to be divided between two subsystems, then Lambda one has a beautiful interpretation. It is the – if it were differentiable, it's the partial derivative of this optimal cost, with respect to that resource. That's what it would be. So if – let's say you're doing minimization and suppose that the – generally, most minimization problems, the actual cost when you really work it out, is dollars. So let's say you're – the units of fi1 and fi2 is dollars and t1 is power, it doesn't matter. I'm dividing power up between two subunits, right. Then Lambda one has a beautiful interpretation. It's literally – it tells you if it's like five, that tells you – let's see if that's right. Would it be five in that case? It would be – sorry; it would be five, and that means that I can do, for every watt of power I'm given, I'll do better by $5 in cost. It's the partial derivative of the optimal cost with respect to the resources allocated.

It tells you the marginal value of power. This one here calculates the same thing and suppose it's three. So if I have two subsystems, you design something, you design something, I allocate power and then I ask if you have to design yourself – please, I don't even ask you to reveal your design. That's x1 and x2, because your competitors and you hate each other. You just happen to be designing subsystems that are going on at the same system. So all you have to do is report back to me what the cost is in dollars of your design, and your marginal utility for power and if you say it's $5 per watt and you say it's $3, then this tells me how I need to shift resources. So, if you can use it more than you can, that's what the five means, at the next step I'm gonna say, okay was that 30, 70? Because now you're gonna get 32 and you're gonna get 68. I don't know if this makes sense, but this is the – I mean, these things, once you think about them – I mean, you have to understand them on many levels. You have to understand them at the convex analysis, convex optimization level, but it's very important to understand these methods also just make complete sense intuitively. They just make sense. By the way, if I overshoot, what'll happen is these will switch and now you need more and then I'll shove back more and that's actually what – that's what these are. That's me going back and

forth, basically shifting microwatts back and forth between you and then at each step, your marginal utility is higher and now, but we're out of the third digit, so actually, in that example this is what we call done and then I'd be shifting power back and fort between you. But at that point, your marginal utilities are equilibrated more or less, and that's how we know we're done. Okay, and this is a resource allocation, so the optimal resource allocation turns out not to be – those add up to zero. Do they or –? Thank you, they're two. Sorry, thank you very much. That's two, thank you. I confused myself. Here there's two resources and then the example I was – the verbal example there was one, so, sorry. These don't have to add up to one. This is, you give .5 to this one and then there's a corresponding down here, the one that looks like it's negative and then this one goes like that. Okay, okay.

So let's do dual decomposition for constraints. That works like this. That's very easy. You simply for a Lagrangean and everything just splits out. It's completely separable. That involves x1, x2 and in this case, these are prices and so again, it works like that. If we go back to our scalar example, you're designing a subsystem, you're designing a subsystem. Instead of allocating you 30 milliwatts and you 70, I now say, make a fictitious internal charge in your design of so many dollars per milliwatt, and so – but use as many as you want. You're gonna use five watts, go ahead, but here's the price, so that's it. Oh, so that's how that works, and then what happens is I publish a price for power, and you guys each design your thing, taking this internal accounting into account, because that's what this is and then, I add up and then report to me how much power you use. If it's more than 100 milliwatts, that's my limit. I crank my power up, because too much power is being used and then presumably I then send back you a message which is a dual update message that says, I've updated my prices, redesign yourself. You do that, and then you come in. I overshot on the price, now together you're using 90 milliwatts and so I now reduce the price a little bit and I keep doing that until between the two of you, you're using 100 milliwatts. That's the price update, that's dual decomposition here. So that's that.

Okay, so here's dual decomposition, and it's quite simple. By the way, this is the vector of resources used, and the + is the violation. So this is a projected subgradient method, because here the prices have to be positive. So this is the projected subgradient method, and it works, and of all the other stuff we've done, it just kind of works, so, and it's actually quite cool. It's a price adjustment method and so on like that. So, and here the coordination is being done, so you can coordinate through prices and you can coordinate through allocation and so some people talk about market economies and controlled economies, and this was very popular in the '60s or '70s or something like that; so, to interpret these things in economics. Okay, so again here's our same example, but in this case, we're adjusting resource prices, and this shows the prices on the two resources and we started them off, I don't know, I guess we started them off up here or whatever, and they shot around. It looks like they're converging to whatever it is, .25 and .35, would've been the right places. So if that were the prices, so then – so the idea is if you get the prices right – I mean, this is sort of the essence of duality. If you get the prices right, then independently, the market will clear, meaning it will use – we will not use more resources than we have, No. 1 and No. 2, you'll actually be optimal. So okay, and this the – you get

the same sort of thing that in a handle of steps, in dual decomposition you get a lower bound and then this gives you the projected feasible allocation here, and you can see here that by this step, you're pretty much done, for all practical purposes. Okay, and again, this is the same – I won't go through this. So, okay.

So now what we're gonna do is we've talked about all of this with two subsystems, which is actually a little bit boring. It's the easiest, it's the simplest thing to look at, these two subsystems and you have to understand everything for two subsystems first, but now we're gonna talk about general decomposition structures, where it's much more interesting, and next lecture we're gonna do a lot of applications in network flow and I forget what other areas we're doing. The most interesting ones is when you have much more general decomposition structures. So, here's what we're gonna do. I should add that there's no real reason to force economical form. So the way I view all of this decomposition stuff, is it consists of a set of ideas, so like primal decomposition, dual decomposition, and it's best just worked out on the fly. So when you have a problem, if you're doing networking or wireless or cross-layer optimization or something like that in communications, or whatever application you're doing this in, it's best to sit down and simply use the ideas, not the actual formulas, right. Maybe the formulas will fit your case, but generally speaking it's better to just do it, to actually take the ideas and derive it all again from scratch in whatever your particular problem is. But what we'll do here, is we actually will take economical form, not because you should do everything by putting it in economical form, but because we'll actually look at some of these higher order ideas about – and actually, it's where decomposition gets really interesting.

Okay, so here's what we're gonna do. We're gonna have multiple subsystems, not just two. We're gonna have variable and or constraint coupling between subsets of systems, so it's not gonna be described by a graph. It's gonna be a hypergraph; so for example, I'll draw a hyperedge if there is a constraint that links a subset of the system, right. So we're gonna have a hypergraph here, not a graph. Okay, and a hypergraph by the way, an undirected hypergraph is nothing but – hyperedge is nothing but a subset of the nodes. So it comes up in a bunch of variables, but that's all it is. It's subsets of nodes. Now actually, what we're gonna do is we're gonna assume is that all coupling is via consistency constraints. In other words, if they're resource sharing, if they're – any kind of equality constraints. We're gonna make them all coupling constraints and the way you do that is simple. If you have like two, if you're sharing your resource, you simply take the copy of the resource constraint, build it locally and then you have to work the consistency. So, without loss of generality, all constraints are consistency, right. So, and it would be something like this. If you and I are sharing 100 milliwatts, right; then you would have a variable, which is how much of that you think you're allowed to use. I would also have a separate copy of that, which is how much I think he's allowed to use. I will then use 100 minus that number and you'll use that number. If they're consistent, it means that we're actually correctly divvying up the power. So everything comes down to consistency, so all we have to do is drive towards consistency.

Okay, so let's do – a simple example would be something like this. It's already more complicated than the ones we've seen. So I have three subsystems here and I have private

variables like x1, x2, x3 and I have a public variable y1, y2, y3 and y4. Okay, so here the private variables are inside these pins and that's to use the electronics analogy. The pins are associated with public variables and an edge actually is – we'll see later what an edge represents, but an edge is gonna represent a constraint, a consistency constraint. Okay, so in this thing you have two simple edges and this structure represents this problem. You have f of x1, y1, so that's the cost function term here. That's the cost of this one, this cost function depends on both this variable and that variable and then the last one is this and then the constraints are, that's constraints local to subsystem one, constraints local to subsystem two, and constraints local to subsystem three, and then they have – I've introduced private copies of the variables and here we insist that y1 should equal y2, and y3 should equal y4. These are the consistency constraints. So without these consistency constraints, everything separates, okay. By the way, the electrical analogy is very interesting and for those who know about electrical engineering and circuit analysis and stuff like that, it's actually quite useful because you could think, and this is again if you don't know about this it's fine, but if you do, you can think of like y1 and y2, if you thought of that as a wire, if you hook a wire between two things, what's equal on the two pins? What potential or the voltage? So you should think of y1 as like a voltage.

Oh, and what can you say about the currents flowing in here and the currents flowing in here? They sum to zero, exactly; so the first one is like ADL and the second one is KCL. Okay, so the currents, in this case you're gonna have to do with the Lagrange multipliers, right, because we're going to expect that. So again, for those of you who don't know about circuits, no problem. Okay, so here's a more complex example, it's quite interesting. You've got like five subsystems here and what happens here is one constraint says that these three variables have to be the same. This one, this one and this one – oh by the way, what can you say about the voltages on these pins in this case? They're equal. And what can you say about the currents entering subsystem one, two and four? They sum to zero, that's [inaudible]. Okay, so we'll get to that. So this is the idea. By the way, this now a quite complicated system, right. It's not a simple thing, it's not given by a block diagonal with little arrow things like that. It's a more complex structure here; interaction structure, that's it. Okay, so the general form is gonna look like this. We're gonna minimize a sum of local objectives, subject to some local constraints, but a local constraint and a local objective can mention two things, it is allowed to mention private variables and it is allowed to mention public interface variables. In fact, the excise don't make any difference at all. They actually just go away because you minimize over them. I just draw them here to remind you that there are private variables, but in fact, you don't have to, you just get rid of the x's. You don't even need them. In fact, at some level, they're just confusing things.

Then you have this, the only thing that couples. This is utterly separable, except for this, consistency constraints, and consistency constraints say this. For every hyperedge I introduce, I'm gonna have a vector of variables, which is in size, equal to the number of hyperedges here and then ei is gonna be a matrix, it's gonna give the net list or hypergraph and it's gonnna tell you each entry, each pin, which hyperedge it's in. So these ei's are matrices whose rows are unit vectors. I mean, this is kind of a silly data structure for it, but still, it's good enough. So, I mean, it's fine, actually. It' just tells you,

it's a note that tells you which – on the y5, the fourth component, which mesh is it in, or which hyperedge it's in. So primal decomposition is gonna look like this. I'm gonna minimize this, I'm gonna fix these things. If you like, I'm gonna fix the voltages, actually, here. I'm gonna fix the voltages and what I'll do is you distribute the net variables to the subsystems, that's what this is. This looks like a matrix multiplier, but it's not, it just distributes the given levels, then you optimize the subsystems separately and you get the – each of these is gonna give you a subgradient here. Then what you do is you collect – if you're working at what the – if you want to get a subgradient of this sum here, you simply do this. You sum – yeah, I transposed gi, it's silly. That's actually – it looks complicated, but in fact it's nothing but summing over the edges. And by the way, this has a – these are the currents flowing into these pins, if you want a circuit analogy. This is the sum of the currents on that subnet. That should be zero, that's KCL. So g is in fact the current, the KCL residual. It's the amount by which KCL doesn't hold. That's what g is, and then it says you update the net variables. So you will update the voltages, like depending on this current. This is the current residual, like that. Again, if you know electrical engineering, that's how that works.

Dual decomposition – and I'll draw actually a picture in a minute that makes this look like a circuit. So dual decomposition looks like this. Instead of fixing the voltages at the pins, I'm gonna actually fix the currents flowing into each pin like this, but the currents are gonna be consistent, so they're gonna be – I'm gonna allocate currents, so when I have a network of three things coming together, I'll say that's +1 amp, -2 and then whatever, let's do +1, here, and they add at zero. So that's the idea, I'm gonna just – I'll make something that's KCL compliant, but now what I need is something like a KVL residual. The KVL residual is – you'll set the currents at the pins, and you'll collect the voltages and if they're all equal, you're in KVL compliance. Actually, in that case, your primal feasible and you're done; but instead you'll actually calculate the average value of the public variable. This looks complicated, it looks like a pseudo inverse, but e is this matrix, it's just got ones on each row. It's a unit vector, so this is just the matrix formula. It is projection; this is a projected subgradient method. Actually, all it really is, is averaging the public variables over each net and then you update the currents that way and I'm gonna draw a picture and we'll see how this works. So this is this bigger, more complicated example and just showing how these things work and I think I won't go into all of this because I guess it's not – I mean, obviously it works, right.

That's kind of – that's obvious, but is kind of cool to actually put these things together and see that they work. I should mention all the source code for this is online. This is actually running, this is a dual decomposition method, and here, this is the residual, this is the total residual, the error in your KVL or whatever, your voltage consistency, or it's just the error in your voltages and that's it. Okay, so what I want I'm gonna do now, is actually, I want to draw the pictures for what these updates are, and then I think that will make it clear. Four people in electrical engineering, it will make very clear. So let me draw that picture. So let's do this, let's have three pins like that and let's say that – let's stick them together like that, okay. So primal decomposition works like this. I fix a voltage on this, I fix a voltage, I enforce a voltage on this net, okay. Now, a voltage says that's basically fixing the value of like y1, y2 and y3. I just fix a common voltage, I say

it's 1.1 volts here and tell each of these to deal with it. So they deal with it and then they will each report a current, let's say new1, new2 and new3, okay. Now, if the sum of these add up to zero, you're actually optimal, the whole thing is optimal.

And by the way, in the circuit case it means you're in equilibrium, okay, so that's what that means. Okay, so instead what happens is I'll add up the currents here and, for example, if they're positive, I will do the following. If these are positive, I will decrease the common voltage, okay, because then less current will flow. I mean, by the way, if you want we can do it both circuits and we can do it in economics. We can do either one; let's do circuits for a while. The circuit says, and in fact, if you want to know what this is, I'm gonna show you what it is. It's that, okay. So, let's work that out of why it's this. Let's take a voltage on – well, that's a wire, the capacitor; there's a common voltage y on all of these things, okay. As a result, a current flows here, here and here and the total current that flows out of the capacitor is minus the sum of these currents. If that's zero, done, everybody's in equilibrium, okay. However, this current – actually, let's propagate forward in time some appropriate small step. If you propagate forward in time, it simply detriments the voltage off of here.

So primal decomposition is basically running this electrical circuit until it comes to equilibrium. When it comes to equilibrium, you will have the following. The voltage here, here and here will be the same. Well sure, because there's a wire connecting it, but more importantly, the sum of these three Lagrange multipliers will be zero, and that's KCL, so for those of you who know electrical engineering, this is kind of the picture. Makes sense? So that' the picture, okay. You can also do it by economics too. Economics basically would go something like this. I'll simply from the headquarters of my firm announce what y is, just say y is this. Each subsystem, deal with it. They each deal with it, but they come back with a number that says how much better they could do if they were given a little bit more of that resource, okay. Now, if the sum of those numbers doesn't add up, it means I could change the amount of resource, I could change the resources and actually do a little bit better, so that's sort of the idea there. Anyway, so this is the primal decomposition. Dual decomposition I'm gonna do in a different way. I'm gonna do it for the tube case, because most people don't know about multiple terminal inductors, so if you don't mind. Now we're gonna do dual decomposition. Dual decomposition is this picture. In dual decomposition, remember, y1 and y2 are not the same. So here's dual decomposition, that's dual decomposition. I claim, and let's see how this works.

Well, what is fixed in dual decomposition is a current like that and it's a positive current for this guy and a negative current for that one. So I distribute a current new and one of these subsystems has got a minus sign and the other one it's a plus sign, okay. Now what happens is they get a current and then they develop a voltage here y1 and y2. If the voltages are equal, we're done, everything is in equilibrium. Not only that, if the voltage across an inductor is zero, then the derivative of the current is zero; so the current is stationary. Otherwise, what will happen is this; y2 might be bigger than y1. In which case, in a small time later, this one will actually decrease. It will actually decrease. It will have new dots; l new dot is y2 – y1. That's the current equation. So in dual

decomposition you're actually integrating forward this system and you're waiting for equilibrium. At equilibrium, you will get a constant current, which is the optimal Lagrange multiplier and the voltage across here and here will be what at equilibrium? Zero, yeah. So that's again a way to do it. So in this case, you're forcing a current and you're waiting for the voltages to equilibrate, and in the other case, which would be in the two terminal thing, this – the voltage is fixed across them, you have primal consistency and you're waiting for the currents to equilibrate, which is your waiting for dual feasibility or something like that. So I don't know if this makes sense, but this actually a very, very good way to think of these things, okay. Okay, so I think this finishes what we –next time we actually are going to do instead of abstract decomposition, we're gonna do specific applications of decomposition methods and things like that; and as I say for those of you who came in after or during my rant or something like that, I should say that we'll be around.

Please get in touch with us about your projects. Some of them were very, very clean, but think about the things I said, or go back and watch my rant and make sure that none of it applies to your project, but come and talk to us because we now know what they're all about and we'll have constructive comments for all of them. So we'll quit here.

[End of Audio]

Duration: 70 minutes