ConvexOptimizationII-Lecture14

Instructor (Stephen Boyd):How's yours going?

Student: We've got lines.

Instructor (Stephen Boyd):What? You've got lines as far as [inaudible]? You did?

Student: No, as far as random.

Instructor (Stephen Boyd):Oh, as far as random, I meant. That's what I meant to say.

Student:[Inaudible].

Instructor (Stephen Boyd):Let me – no, let me find that. Are you guys gonna tell me when we're on because I've noticed the last couple of things – we're on now? All right. Well, good thing I didn't say anything I shouldn't have, not that that was happening. Sorry, we'll continue that discussion later. All right.

Well, a couple of announcements. Let's see. Oh, the first one is right after class today I'll have office hours. I feel guilty since I was in New Mexico last Tuesday. And actually, as it happens, I'll be in Zurich next Tuesday. So and, in fact, we're gonna tape ahead next Tuesday's lecture tomorrow.

I think it's like 12:50 to 2:05 here. So I feel like I might as well just bring in like a cot and stay in this room because I don't know. It would just be easier, right? A little espresso machine? Anyway, so tomorrow 12:50 to 2:05, that's on the website.

The other announcement I wanted to make was that the midterm progress reports or something for the projects are due tomorrow, so and do feel free with any of those to wave them under in front of me or either TA just so we can – because we can quickly look at it, scan it, and say, "Take it away," or point to things because we – I actually think really the more feedback you get, the better instead of just sort of – I don't know.

Basically, we don't want to read stuff we don't like. That's sort of – so I'd rather look at it for 30 seconds, catch a problem instantly, and just say, "Take it away and have it come back later." Anyway, so those are due tomorrow. Let's see. I can't think of any other sort of administrative details.

I guess you have a homework due today, and we're gonna make up another one and put it out like, I don't know, maybe today or tomorrow or something like that. There will probably be only one more homework after that one, so, and the idea is we'll just have like one exercise or something on each of the topics we're gonna cover just so that if we cover something you can't say you didn't do a problem on it.

So yes, you’ll run a CG. You’ll have to run a CG thing. We’re trying to arrange these to be incredibly short so that you write ten lines of code simply so that you can say, “Yes, I’ve solved a problem with 100,000 variables in .2 seconds or something like that.” Just so you can say you’ve done it.

Let’s see. The other – oh, and another thing I was gonna say was that we’re gonna settle on the last week. Some evening the last week we’re gonna have a poster session for the projects. So the projects are due the Friday before the last week. I think that’s May 30th. Is that true? My God, I think that’s true actually because that’s – isn’t that like two weeks? Oh, my God. What happened?

Okay, so anyway, that’s when the projects are due. You know, if there’s any sort of emergencies, I mean we made it intentionally a week before the end of the quarter, so there’s a little bit of slack there, not a lot because I’ll be disappearing soon after the quarter finishes. But we will have a poster session the last week of classes. That’s like June, roughly June 2. I don’t remember exactly when it is, but we’re gonna have a poster session.

So, and it’s gonna be one evening. It’ll either be like the Monday, Tuesday, or Wednesday, and we’ll figure that out, and you’ll hear more about that. Actually, if there’s – if anyone knows of some reason why they can’t – if there’s some other event one of those evenings, someone should let us know quickly before we schedule it. But we’re gonna schedule it. We’ll get poster boards. Well, we’ll get the posters from the department.

I asked the department if they would provide dinner for us. You can guess what the answer was, so maybe we’ll figure out some way to cover dinner or something like that, or something like that for one of those evenings. So anyway, just plan that last week, the first couple of days in the last week we’re gonna have a poster session, and it’ll be just the standard 12. We’ll put more stuff on the website, but it shouldn’t be a surprise. I think all of you know what a poster is anyway. So it’s the canonical form. It’ll be 12 slides.

Your project should not be that long, so, and in fact, if they are organized, if the expedition is organized, it should take you like no time to go from a six page final project report to a poster. In fact, it should be – if it’s organized, the sections are right, you state sort of what it’s about, what the problem is, what are the attributes of the problem, it should just go perfectly. So anyway, you’ll hear more about that.

Okay, so I think that’s all I have for administrative stuff, and I guess we’ll finish up this topic of truncated Newton methods, and then we’ll go on to truncated Newton interior point methods. So we’ll finish up another chunk of the class today, which is – and then we’ll move on to yet another chunk. The chunk we’re gonna finish up today is basically how to solve extremely large problems, so okay.

So let’s look at – let’s go back to truncated Newton methods. So remember what truncated Newton method is. It’s you form a – you’re minimizing a smooth, non-linear convex function, and you form the Newton system, but instead of solving for the search direction exactly by, for example, a direct method that would do some kind of direct factorization or something like that. Instead of doing that, what you'll do is use CG or PCG, preconditioned CG, to approximately solve the Newton system.

Now the way this is done is a bit surprising. What you don't do is actually spend all this – you don't do enough CG iterations to actually get a good approximate direct solution of the Newton system because there's no point because, in fact, your goal is not to solve the Newton system. Your goal is to minimize this function.

So, in fact, these things work really well, and, in fact, they work well precisely when a reasonably good search direction emerges after a shockingly few number of CG steps. So that's generally how these things work well. Okay, so this is an example. I think it's a logistic progression example with LS regularization. It really doesn't matter because it's kind of similar no matter what problem it is.

So this is what we'll do. The problem was scaled in such a way that we could pull off a Cholesky factorization. I think the Cholesky factor had something like 30 million nonzeros. So it'll take some time to do both the Cholesky factorization and also to do the backward and forward substitution. So but direct is possible. All we have to do with this problem is scale it by a factor of ten and direct becomes kind of out of the question, so then at least on a little standard machine.

But we'll compare that with the truncated Newton method where we do several things here. What we're gonna do is we're gonna max – we're gonna simply do ten steps. Now this says we'll exit if we actually do solve the Newton system within .01 percent. Is that right? Yeah, .01 percent, or if we hit ten iterations, so almost always this is gonna be ten iterations.

The second one essentially does 50 iterations, although we will terminate early if we solve the Newton system. And the third one will do 250 steps. Now I guess in the problem I forget how many variables there are, but I think it's tens of thousands or something like that. I actually don't remember. Well, here we go. Yeah, it's on the order of tens of thousands, okay. So maybe it is 10,000.

So it's 10,000 is the size of N, so the – let's bear in mind that the theory say that if you do 10,000 CG steps you will get the Newton direction. So by the way, the practice does not conform to the theory. The theory tells you to get the exact Newton step if you did 10,000 steps in perfect arithmetic, in exact arithmetic, which of course we don't. All these are done in doubles obviously. So but the point is this number is shockingly shy of the number that theory tells you will do the trick, and these are like a big joke, right? That's one-thousandth of the number of steps required.

Now on the other hand if you remember all this business about the spectrum, that's gonna be important. So the question is really the 10,000 says that no matter what the right-hand side is for any A, you're gonna get the exact solution. But the point is in ten steps you'll get a reasonable solution provided certain things happen. And if the spectrum is clustered, if the right-hand side, which is the gradient lies in the – a lot of it lies in the tenth [inaudible] subspace. That'll do the trick. Several things will work, okay.

So here's how it works. Here are the iterations, and let's see, here's the Newton method, which does what we all would expect it to do, which is basically nail this thing in 16 steps to some accuracy way beyond anything you need. So that's our friend Newton here. By the way, that's Newton applied to a smooth convex function, so Newton doesn't work this way in general, but okay.

If you do 50 steps, let's see. Oh, the CG 250 is right on top of it here. So if each of these – what that says is that the quality of search direction you get in 250 CG steps is whatever that search direction is, although no one would claim it is the Newton direction. Whatever it is, it is obviously good enough to provide – exactly match the performance of Newton method, right?

So this alone is gonna beat Newton's method insanely, both in memory and in time. But the really interesting part is if you do 50 CG steps you can see you're not – you know, you're down here, it's charging you one more, so that's even more shocking because that is now one-fifth the effort of this one. And the real shocker is this one where you do just ten steps. The ten step one actually, if you truncate around here, which is actually probably a perfectly good accuracy, is actually quite – is amazingly good. I mean this is actually sort of amazing up here.

Okay, now the real measure in a method like this is absolutely not the number of iterations. That's totally irrelevant. What's relevant is the number of CG steps because the CG step is what makes – it makes a call to a Hessian multiply method, right? And I mean exactly what I said.

You generally don't form the Hessian and then multiply it by a matrix. You're welcomed to do that. There's cases where you would do that, but remember that the interface of a truncated Newton method to the problem is simply you need something that will multiply the Hessian by a vector period. That's all you need. You don't need anything else.

So in this case, and this, by the way, would correspond extremely well with this time. So this tells you how fast it is. And you can see something shocking that ten steps of CG here gives you – if that's a good enough accuracy for you, it does this in just a shockingly small number of steps. So that's CG.

This is very common. This is – I guess it's called – sort of on the streets it's called jamming. The theory would tell you that if you did things in infinite precision, which you don't, that this would eventually converge. Of course, it might take a long, long, long time or something like that. We don't do things in infinite precision. We do things in doubles, and so in fact, it might just state here. In any case, for all practical matters, this is just – you're just wasting your time out here.

Yeah, by the way, this is gonna be a number so small that it – we're talking – well, we'll see in a minute what the ratio, how much faster this is than a direct method. So you see actually everything you see that is typical. This is quite typical to get extremely good conversions. The width of each of these guys is ten steps. The width of these is 50. They could be a little shorter than 50. They would be a little shorter than 50 if basically you solve the Newton direction.

Now what I should have done is I should have shown what happens if you do CG1. I don't know why I didn't put it on the spot, but CG1 is basically a scaled – it's not even a scaled gradient method. It's a gradient method. And CG1 actually goes like this, maybe goes like that and then stops like that, okay, so which is to say it's quite useless, okay.

So here – I won't go – we looked at this last time, but these will just give you some numbers here, but the ratios are pretty good. There are things like speed-ups of 400 to 1 here. And let's actually see what happens if you put PCG in. PCG is what you need – a preconditioner is what you need to sort of eliminate that jamming type thing. And so if you put in a preconditioner, what happens, it's not too surprising, is now your CG thing, despite the fact that you're doing ten CG steps, only ten CG steps.

You're actually getting a more than good enough search direction here to get extremely good performance in some just shocking number of steps. By the way, that's whatever is has, like 100 and something steps. Each step is basically a matrix vector multiply which you don't even implement as a matrix vector multiply in most cases. In fact, that's the whole point. So this is just a shocking number of – a shockingly small number of steps to do this.

Okay, so and if you want to see sort of what the numbers are there, they're not bad. They're things like 400 to 1 or something like that. These are good speed-ups. By the way, here the only thing we did was we didn't really want to wait more than an hour or something here, so that's the only – we set the problem size so that this would be like an hour.

The same problem, which I guess you have all the source code for because it's on the website. I promise you you could go in, make it ten times bigger, and the CG stuff will run just fine, so and then you'll be solving problems with a couple hundred thousand variables. You can almost certainly take it to a million and it'll just run. And it'll probably take – maybe it won't take 200 PCG passes, but it'll take something like 300 or something like that, something just amazingly modest, and it will work.

So okay, so there's a lot of extensions here. You can do the things with equality constraints. If you use an infeasible start Newton method but equality constraints, you have to – your equality constraints are actually not gonna be guaranteed to hold, right, because that's a property of – if you take a step of one in a Newton method, your equality constraints hold. In general, if you do a truncated Newton, it's gonna just simply converge to zero.

And, of course, you can take this idea and put it into a barrier method or your favorite or more sophisticated primal dual methods and so on. And I should also say that the distinction between direct and indirect methods is a nice gray boundary because in fact the ones you've been using, STPT3 and Sedume, when they – they use nominally direct methods. In effect, you can even see the right-hand column when you run CBX you'll see – we didn't run. I mean this is someone else's code, but in all of those things, they'll actually do several steps of iterative refinement.

What that means if they'll do a Cholesky factorization, but they will presume that there are numerical errors in it, and they'll simply use that as the preconditioner, and they'll do like two or three steps. So there's a sense in which you've actually already been using CG, but at the other extreme.

This is where you calculate an extremely good approximation of the solution and you have an amazingly good preconditioner. The things I've been showing you are at the other end where you are computing. You're doing a really bad job of solving a gross approximation of the Newton direction, but it's good enough to make progress overall.

Okay, so the last thing we're gonna do is just apply these ideas to interior point methods. So what's gonna happen, I mean you're gonna do – it's gonna be tricky to do this, but it can work well, and that requires luck, tuning, and a good preconditioner. So those are actually generally bad qualities for an algorithm to have, right. Imagine if every time you called or [inaudible] or A backslash B, there were 14 parameters you had to adjust, and you had to say, "Oh, A backslash B is not working." And someone says, "No, I know how to do it."

You have to go back and set A a little bit higher for problems like this, and then watch it, and it runs for a while and then jams, and then you have to come back and you go, "No, restart it and set alpha to .01 and see what happens." I mean so these are bad attributes. Actually, they're good attributes from an employment perspective because it means whoever needs to run these things is gonna need to pay someone to be there to baby-sit it.

So these large-scale methods, they will need tuning at a minimum. Once they're tuned, they may not need babysitting and stuff like that, but they're – so just to make it clear, these are not good attributes of methods. On the other hand, you know, you shouldn't complain if you're solving a problem with 10 million variables or 100 million variables. So that was – you made the mistake by going into a field where such big problems had to be solved. So you can hardly complain them that you have to tune and baby. And the generic software off the web doesn't just work for you.

So okay, so we'll look at network break control. This we've seen a whole bunch of times, and the problem is this. You want to maximize a logarithmic utility. So that's maximize the product of a bunch of flow rates. I have a matrix R, which has got 01 entries, and it tells you if you sum, RF calculates the sum on F. Each row of R actually gives you the links. It's got ones in the places where that flow goes over that link here.

So and then that’s a capacity. So this thing is the total traffic. It’s a vector of the traffic on each link, and that’s the capacity on the link. And so the idea is a whole bunch of flows are sharing resources. They’re sharing the links. The constraint is the capacity on the links, and we want to work out the sharing in such a way that this utility is maximized. This is the negative utility here. So that’s the problem. It’s sort of a classical. Actually, we’ve already seen this problem for distributed methods. We’ve seen that.

The dual rate control problem looks like this. We’ve also already seen this. You maximize N minus C transpose lambda plus, and then you get another log term here. And that’s subject to lambda positive. Now remember the distributive method, so distributed rate control works like this. It solves this dual problem in a distributed way using a – well, using a projected subgradient method, although, of course, the objective is differentiable, so it’s really a projected gradient method although that means something else, and it’s not the algorithm we looked at before.

And the duality gap is simply given by this. If you have F – if you have both a primal feasible flow and if you have a dual feasible lambda, then this is the duality gap. So the primal dual search direction, if you just worked out what it is, it doesn’t matter because all of these search directions end up looking the same. So if you use a primal barrier, dual barrier, all of these things, if you apply a Newton method, they all look the same. They might be a little bit different and everything, but they all have the same components.

Actually, they all look like this, a diagonal plus R transposed D2 R, just period, and that’s for all methods. This is gonna be for primal dual method. What will happen is the D’s might change a little bit, the right-hand side might change a little bit and so on, but otherwise they’re gonna be the same.

All right. So once you solve this to get delta F. This is a block elimination method, and then you get delta lambda this way. And then these are just the details for this primal dual search direction. You don’t have to. You can just as well use a – and you can just use a primal method. It makes no difference.

The primal dual residual is this. You break it into two parts. That’s the dual residual and that’s the centering residual, and then the algorithm just looks like this. This is the standard one. If you solve this exactly, it would be a primal dual method, and you’d expect it to converge in whatever, 30, the usual, 50 steps. But instead, we’re gonna solve this system here approximately using PCG, and then you do a line search on the norm of the residual, which is standard, and so on.

Okay, so here’s a problem instance with, let’s see, 200,000 links and 100,000 flows. So that’s a reasonable sized problem. And there’s about – each flow has a length of about 12, so each flow goes over 12 hops, 12 links in the network, and each link has got about 6 flows on average going through it. We made the capacities random from over a range of 10 to 1, from .1 to 1.

Here the truncated Newton method is done with the following. You stop at either 10 percent accuracy, or that's the approximate duality gap divided by the dual function level, and you have an end – this actually, in this case, end max was never reached. So by the way, you might wonder where this is. This is kind of – when you enter into how to make these things work, how to set these things, there's a lot of lore, so it's a lot of room for personal expression.

So when you use these methods all the art comes in in figuring out a good stopping criterion for CG. And the idea is completely obvious. What you want to do is you want to do the smallest number of CG steps. You want to do the crappiest job of calculating a search direction that you can that'll make the overall master algorithm work reasonably well.

So and you always start by looking at smaller problems and doing lots of CG steps because if you can't make that work, then you're in trouble, big trouble. So once that's working, then you start backing off and start pushing things to the limit. And then only later when you dare do you get down to things like numbers of CG steps like ten and stuff like that.

Okay, this is a diagonal preconditioner. It uses warm start, meaning that it starts – it actually starts obviously from the last search direction, and then these are the parameters here. So here we're gonna go to 10 to the -4 [inaudible] which really isn't needed, but anyway. And here it is, so these are minute U of F. So these are the primal values and the dual values coming up like this.

And what's really ridiculous about this is in 100 PCG steps you basically have a very good – you basically have optimized the flows. You certainly optimized them within, you know, in a couple of hundred CG steps. Each CG step, by the way, is actually a forward. It's actually you multiply by R and R transposed. Those have physical meanings. You know, one sums for each flow, it sums some numbers across the flow, and then R transposed simply calculates the traffic on an edge. So it's literally 200 passes and it's all over. So you – 200 passes up and down and it's globally optimized.

So okay, this shows you the duality gap evolution here, and you can see actually that it's actually slowing down. Of course, if you do a primal dual method you expect to see – with exact Newton you expect to see something that looks like that. This is doing the opposite. It's slowing down.

By the way, I think this duality gap is already pretty small. The reason is this is a total duality gap, what you should really report is the duality gap per flow, and there's 100,000 flows roughly. So this is actually like 10 to the -8 here already or something like that. That's the duality gap per flow, meaning that I'm not even sure why we're showing this, but it was kind of all over about right here, so but okay.

Here's the same problem with a million edges and 2 million flows. So these are real problems. These are big problems now. These are not – you can't type this into CBX or something like that, and they're big, so and that's the same thing. It's just to show that it actually works and so on, and here's the duality gap, and again, I guess it's slowing down or something like that, but it's good enough. This is completely ridiculous, right, the idea that you've got an extremely good solution, provably so in 200 passes up and down across the network. I mean that's really quite ridiculous.

So okay, that finishes up this. I should say a little bit about this. I mean there's a lot of art on this, and if you look, if you just go to Google and type in any of these types of strings, other names would be limited memory, BFGS, and it's got all sorts of names, truncated Newton, iterative Newton.

If you type in any of these things, you'll find you'll start looking at some of the lower. And sometimes they flatly contradict each other. They'll be – one book will say you should make your stopping correction for PCG's go this way, and another one says it should grow. I mean so clearly no one has any idea, and you just deal with each problem on its own, so okay.

I should say one other thing is that I personally am no expert on large-scale optimization, but just for fun a couple of years ago we decided we would do some just to see how it works. And I can say the following. I am reasonably comfortable saying now I don't know of a problem where – we've never failed, ever.

So we did a couple of machine learning problems, some finance problems, some control problems, did some – oh, let's see what else did we do? Circuit sizing and stuff like that. Yeah, the details differed every time. Took one grad student week, that's a unit of time, a unit of effort, I should say, to tune things, but no problem. I would just come in one day and some grad student, when I would come in in the morning, they'd be leaving, and they'd say, "Good news. This PCG update rule nails it across a huge number of samples."

And so anyway, so I think the bottom line on this is pretty simple to say. It's just this. If you have a problem that's in the million variable, 10 million, of course, all of this, if you want to go much higher, no problem. But everything's gonna come down to your ability to do this matrix, to actually – well, to multiply it by the Hessian.

So you want to get a huge super gigantic problem, and you want to take 1,000 of your closest friend's processors, and make some giant NPI thing. I don't' see any reason why it won't work. I just don't see why it won't go to 100 million variables or something like that.

So then it's nice to know that if you have a problem and you solve small instances of it with CVX, and you like what you see, and it looks good and promising, it's nice to know there's actually a path to full large-scale things. So and I think that's really the takeaway message here.

Okay, this finishes up one whole section of the course. It is on solving giant problems on a single machine, although it doesn't have to be a single machine, in a sort of centralized way. It is actually centralized because the CG algorithm is centralized.

There's inner products are the parts that are centralized, and but I don't think it's that big a deal if – I mean if, I don't know, someone wanted to do this. It's not a big deal. These are just – they're scatter gather operations. And if you're only doing a couple hundred, I don't see why you couldn't do this for like really, really big problems.

So I do know someone who solved a billion variable LP just for fun, so not using these methods, used direct methods, but it took an entire – it took one of these blue jean, you know, it's an entire room full of computers. And each iteration, I think, took something – it was either like an hour or two hours per iteration. And I asked him how many, you know, what algorithm he used. Same one, primal dual, everybody else's. How many steps did it take? And he goes, "Same, 21." So but that's like a day and a half or whatever it was, so anyway.

So I think it's no reason to believe that these things won't go to much, much larger things. If you need to go to these large systems, I think it will work. Okay, any questions about this stuff? Like I said, we'll arrange that you will have person experience with CG, so and we'll arrange for your personal experience to involve nothing more than just a handful of lines. But you'll do CG, you'll run it, you'll solve something big, and then you can say you solved CG, or you solved the giant system using an iterative method.

I'm just curious actually. How many people have seen this stuff in other context? In where did you see it?

Student:[Inaudible].

Instructor (Stephen Boyd):Oh, sure, yeah. That's like whole courses, right?

Student: Yeah.

Instructor (Stephen Boyd):Okay.

Student: Finance.

Instructor (Stephen Boyd):In finance, really? Which class?

Student:[Inaudible].

Instructor (Stephen Boyd):Oh, okay, at KTH.

Student:[Inaudible].

Instructor (Stephen Boyd):Oh, [inaudible], sorry. I'm mixing up my Swedish students. Really? So CG for truncated Newton and just CG or just CG?

Student:DFGS.

Instructor (Stephen Boyd):DFGS, oh, okay, for some class on finance. That's cool. Okay, all right. The rest of you should just know about it. So maybe you'll never use it, but you'll do one, and then some day five years from now there'll be some giant problem and you'll be banging your head, and you'll look up and there will be a big light bulb and it will say in the middle CG. And you'll get to use it for something. You'll get to use it for something, sure, okay. All right.

Next, now we're gonna do – another chunk of the class is gonna be on a – I mean it's a family of methods that's been around for a long time. These things go back into the 50s, and someday when I'm bored, I'm gonna go back and actually find some of these things, these references. But it's like a lot of other things like interior point methods as well.

What happens is that for unknown reasons relating to marketing and just who knows what, fashions. All of the sudden, even though some exist for 30 years, the right person goes to the right place and somehow each person there tells ten of their friends, and then they each tell ten of their friends. And this happens three more times, and all of the sudden, this thing bursts on the scene as if it was brand new.

So that's the case for L1 norm methods for scarcity. You should know that it is in super duper fashion right now in many fields, in statistics, in machine learning, it goes on. And, in fact, in some cases, they're so damn sick of it in like machine learning, someone actually described – told me, he said, "No, we're in our post L1 norm stage," meaning that they're so sick of it and everybody's done it, and they've already had their 300 papers on it at the last Nips or whatever it was. And so they're actually just sort of moving beyond it, which is fine.

Okay. So and I'll say more about the history as we kind of get into it. So, and it's got different – oh, the other thing to warn you about is it's got different names. I don't know if they're even in the slides here. The names, you're gonna hear about it, are – some people just call them L1, L1 norm tricks. Let's see. There's another – you'll find in statistics you get lots of names. I don't know why. You get lasso. Let's see, basis pursuit, several others.

Oh, in machine learning, you get things like SVM is support vector machine. People who do this will never admit that these are just L1 tricks. Trust me, they are. So they'll never – yes, and I hope some of you out there are watching this online and getting pissed off. So that's exactly why I said it if you're wondering, because I don't – there's no one to piss off here.

Anyway, so support vector machines, and let's see. Oh, it's used in actually geophysics actually I think is one of the earliest examples I can find. That's right from Stanford.

That’s 70’s. That’s [inaudible]. And there’s a rumor of, and I’ve heard something about some papers on antenna design from the 40s. I’m gonna – I have to hunt those down using these methods, so okay, all right.

So let’s go over this. These are – they all involve problems with cardinality. So now cardinality is just the number of nonzeros. Well, of course, it’s the size of the set, but the cardinality function on a vector is simply the number of nonzeros in it. So these come up a fair amount. They’re actually hard to solve exactly. A lot of them are MP hard. Some of them, people have shown that, others not. Some might be easy, but there’s a very limited number that are easy, the simple mapsac problems and things like that.

But the vast majority of these things are hard.

Okay, so the other interesting thing about it is that it’s simply [inaudible]. It relies on the L1 norm. Oh, I forgot to mention my list of fancy names. Compressed sensing is one that’s making the rounds in statistics and applied mathematics, so you’ll hear that. And maybe there’s at least one or two other – it’s got lots of names, but as I think of them. Oh, here’s some right here, okay.

So it’s been used for a long time. In engineering design, it’s been used for a while, and we’ll see where that comes up. It’s been used in statistics. Again, there’s a Stanford connection because the originators of this are at Stanford. Let’s see. Oh, it’s also related to [inaudible] that goes in the 70s. Support vector machine and machine learning. Oh, this is from the maybe 80s or even maybe early 80s, this total variation reconstruction signal processing and geophysics.

This again, like support vector machine, do not, if you see a person who does total variation reconstruction, don’t say, “Yeah, that.” Actually, do say, “Oh, yeah, yeah, sure. That’s just an L1 trick,” because it will really piss them off. Okay, and compress sensing, that’s the newest import. Okay, there are some new theoretical results. They are very cool because it really didn’t occur to anyone that you could actually say anything about these methods. Those are very cool. I’ll mention them, but needless to say, that’s not gonna be our focus.

Okay, so cardinality of vector, it’s actually a separable function. So it’s a sum of – it’s this. It’s actually the sum over all XI's of card of XI. But card of XI is nothing but zero or one, depending on whether the variable is nonzero or not. That’s quasi concave on R plus to the N, but not RN, for what that’s worth, which is not much because that would tell you that if you wanted to maximize the number of nonzeros, you could do it of some positive things, but that’s kind of silly.

Otherwise there’s really no convexity properties, and it arises in lots of problems. And let me just – I mean I know this is kind of obvious, but I’m just gonna draw the picture of the function just so we all know what it looks like. It looks like this. Here’s X, and it looks like this. It’s that and then that. So this is pretty non-convex, I think you’d have to say. Actually, we'll leave that picture up because it's gonna – the whole thing's gonna be highly implausible, I mean why these things work.

Okay, so convex-cardinality problem, it's a problem that would be convex, except cardinality comes into the problem somewhere. And it might come in the objective, but it also could come in a constraint. And these are really interesting. This says – these are very interesting problems, so is this one. This says the following. This says, "Find me the sparsest vector that lies in a convex set." Boy, are there a lot of applications of that.

For example, C could be the set of coefficient vectors that explain your data within some reasonable level of confidence, okay. So these are the – then this says, "Fine me the simplest description explanation. Find me the simplest model." So that's what this is. So this is even – that would be like [inaudible] or something like that.

It comes up in engineering design as well. So here, for example, the Xs are often things like widths or something of wires, let's say in a circuit. And then if a wire has zero width, it means the wire isn't there. So it's basically saying, "Make me a power supply distribution network or something, or a clock distribution network. Obviously, you can't give a tree because if you get a tree, you can't remove any wire segment, so you get a mesh.

And then you're basically saying, "Find me the sparsest thing that will actually this thing that will actually satisfy all the specifications, okay. This comes up lots of places. Another good example would be design me an FIR filter, and these are the coefficients in the filter.

What's cool is if I have a filter and it only has something like 14 nonzero coefficients, zero coefficients, I don't have to – I could build custom hardware that will just have 14 multipliers, I mean, or that could be base or time, but you get the idea. You can make very, very – as far as FIR filters, you get implemented hardware very quickly. Actually, software as well if you do it the right way. So all right, it comes up in lots of cases. We'll talk about it in all of them.

This one says, "Choose me no more than K of these coefficients." Okay, so how do you solve these problems? Well, the first thing is this. If you fix the sparcity pattern of X, okay, of which there are two to the N. So to fix a sparcity pattern says that for each component, it's either nonzero or zero. There's two to the N choices like that. One choice is, of course, X equals zero. The other choice is X is full, so all things are there.

There's two to the N sparcity patterns, but if you fix a sparcity pattern, you get a convex problem and it's all over. Now what this says is that if N is, let's say, ten, these cardinality problems can be solved immediately and globally because it just solved one or two for convex problems and you're done. Everybody see what I'm saying?

And, in fact, there's tricks there, you can even do it faster than that if you know what you're doing because, for example, you can do things like this. Once here, suppose we want to find the – minimize the cardinality. The sparsest X is in some set. This might be some design. Once I have found, if I have ever found an X with a cardinality, say, of eight, I never have to look at – obviously there'd be no reason for me to look at any sparcity pattern that has more than eight, right.

So you can often trim a large number of these things, even if you're doing direct enumeration. But you can simply do direct enumeration immediately if there are, for example, just ten variables, or 20. Twenty is a million. That's overnight or something, or in C, it could be some reasonable amount of time. Okay, all right.

Now the general problem is NP hard. That's actually quite easy to show, and it's not that relevant either. Now you can solve these methods globally by branch and bound method. We're gonna do that later in the class. These are global optimization methods, and these things, well, they can work for a particular problem. And these are gonna be – this is gonna be the global solution.

But in the worst case, these things reduce to checking all of the sparcity patterns, or at least many of them or something like that. So these methods are actually heuristic. So there's a very good way to write these out as a Boolean LP, a mixed – and there's a way to write these out. So a Boolean LP looks like this. You minimize a linear function subject to linear inequalities. But these Xs are simply zero one.

Now this problem here, you can embed essentially any combinatorial problem as a Boolean LP, any with a reasonable embedding. I mean not one that grows size or anything like that. So you take your favorite three traveling salesmen, and I'll easily write it out this way. Now if you want to see how to do it, it's actually pretty straightforward. What you do is this, is you write down this. You write minimize C transpose X, X less than B. And then you write the cardinality of X plus the cardinality of one minus X is less than N, okay, less than or equal to N.

Now what' interesting about that is the only way that will work is that the XI's either, for each entry, either this one is zero, either that zero or that zero, and that's the same as saying X is either zero or one. So it's a very quick embedding, okay. I mean not that it matters much.

Okay, so now we're gonna talk about applications. I already hinted at this, but here's one. Oh, and by the way, I think these methods have not diffused anywhere near the extent to which they should have, right, because what's happening is a lot of work on this. You know, some people know it, but they know it in a weird way. Like if they're in statistics, they just do SVM's or something or L1 regularize this or that.

They haven't really internalized this. It's just a completely general heuristic, powerful heuristic for getting sparse things period. And there's lots of people poking around who could use these things and have no idea about these matters. They go to Google.

The next thing they know, they're reading some fantastically complicated math paper with a result saying that if you do this in some incredibly special case and something's bigger than something else, then you'll actually get the global minimum with overwhelming probability or something like that. And that's not – what they needed in fact was simply an excellent heuristic for solving sparse problems or something like that. So anyway, all right.

The sparse design is this. This is the set of vectors that satisfy a bunch of specifications, and you want the sparsest design and so you can have FIR filters. Like I said, one of the first applications, this was antenna array beamforming. And here the zero coefficients correspond to the unneeded antenna elements.

So truss design is another one that's actually very – this is actually not only – this is actually now has huge commercial impact, so there's a whole industry that does this kind of stuff. And it's not just trusses, but also to mechanical stuff. And here's zero [inaudible] bars that are not needed. Wire sizing, I already mentioned that.

Now what's interesting about these applications is, for example, let's take the antenna array design thing. So what you do is you start with you say, "Here's where I'm gonna put my antennas on something." And you start with 1,000, and someone says, "Well, you can't possibly put 1,000 antenna elements there," because first of all, we can't afford the – you can't afford that many. They won't even fit physically. You can't even fit them all there.

The second is you have to have an RF amplifier for each one and blah, blah, blah. It goes on and on. And then you say, "No, no, no, no, these thousand antenna array elements, you shouldn't think of them as possible antennas." I have absolutely no intention of using 1,000. In fact, I intend to use, let's say, 17. So then you say, "Well, then why are you using 1,000?" And you say, "Ah, because this method is gonna pick which 17 I'm gonna use."

And the same in truss design. You make something and you put 10,000 trusses or 20,000 bars in the truss. You have no idea – you have no intention of building a truss that's 20,000 bars or 200,000 bars. You're gonna let this thing decide on the 132 bars it's gonna use, and that's a reasonable truss.

So okay, so this is kind of obvious. In statistics and modeling, it's completely pervasive, absolutely pervasive. I mean, and it's completely fundamental as well. It basically says the following. Oh, this would be a simple one. This says, "Choose K columns of A to get the best fit with B." So, and that's a cape term model, and again, here you could do this by simply enumerating all N choose K choices. This is quickly gets out of hand if K is more than a handful and is big and so on and so forth.

The variations would be find me the – choose me the minimum number of columns which allows me to get a good fit. You could put it this way. I mean there's lots of variations. This is progressive selection. By the way, this could be quite interesting because the columns of A here actually are regressors or features. And so you're actually doing – this is in other people and say, "This is what we call the machine. We even cause feature selection."

So this would be feature. And you could throw in all – so what this means is actually kind of turns a basic principle in modeling on its head. So let me exaggerate here, but say that basically as of ten years ago or before these methods were proliferated, the basic story in modeling goes like this. You say, "I want to predict the next term in this sequence. I have this sequence here." And someone says, "Fine."

You construct a model, and you construct something like autoregressive model. And you say, "I'm gonna make a linear combination of the last ten entries or something," or it could be the last ten entries plus weird quadratic terms plus sick things that your domain experts throw in there, like they say, "Oh, here's an interesting feature. How about this?" The highest value in the last five steps, you know. Who's to say? How about the spread? The largest minus the smallest value? How about some weird time weighted maximum? I mean you can go nuts.

By the way, this is all good from an employment point of view for the experts, right, because if it takes the experts to generate all these features, that's great. But then you say, "No, no, wait, wait." Oh, obviously the more features you add, the better you can fit your data. So if I give you a pile of data and if I make – if I give you 100 data points and I have like 100 features, then generically, I'll be able to fit it exactly and say, "Wow, I'm doing really well."

That's stupid. It will have no generalization ability, and you would verify that by cross validation. So you'd take your data, have two sets. You could even be formal about it, not even allow the person developing the model even access to the other one, and then you'd try out this way and you'd find over fit. So the problem is you need the expert to choose a small number of features. Everybody following this story?

So this is modeling, statistics, whatever you want to call it as of 15 years ago. The story is everything comes down to choosing the right features. You get the right – and a small number of them too, by the way. So notice that puts the burden on the modeler, so that's the idea.

And then how would they do it? They'd sit around at home or whatever. They'd actually have their grad students do it more likely. And they would try various combinations of things, and someone would say, "Oh, my God. I got an amazing fit." It turns out that the volume minus the maximum minus the spread, blah, blah, blah. If I throw in these six features, I get an extremely good prediction of this. Everybody see what we're saying here?

Okay, all right. So this completely turns it on its head. This basically says you can do feature selection, and what's super cool about this is it could be fat. If you have a fat A matrix in a general regression problem, that's just stupid. That's you're automatically over fit. You’re way over fit. There’ll be lots of combinations that will reconstruct your data with no – zero error. They will have no generalization ability, but so that is totally uncool in a fitting problem, okay.

So now here it works fine. What you do is you get – you have to have something like 100 measurements, and you give 10,000 features. So not only are you doing a bad thing, but you’re doing it by a factor of 100 to 1. Everybody see what I’m saying? You’re so over fit, it’s ridiculous.

But what you do is you solve this problem and you either put a cardinality or you solve one of these problems here. And what those 10,000 features are, you have no intention to use 10,000 features to fit your 100 data points because you could fit them in a huge 9,900 dimensional null space of ways. You could fit them exactly.

So what you’re actually saying here is the following. These are 10,000 features. Most of them are probably completely irrelevant, and in any case, I’m not authorizing you to use 10,000. You may choose 30 of the 10,000 features, and that’s exactly what this problem would do. So by the way, the implications of this, I think are gonna ripple for the next ten years. It’s gonna completely change modeling and a lot of other stuff. Everybody see what I’m saying here?

So this is very – I mean normally people talk about this is a much lower level or everybody knows all these things, but I actually haven’t heard people say it this explicitly. But everybody got this here? So you can do wild. I mean so, by the way, it’s interesting. You got to totally change your thinking. If someone says, for example, “I want to fit this data with a polynomial of these things,” and someone would say, “Well, that’s a pain in the ass, and it’s not gonna work well,” because if you have ten variables you want to fit a polynomial, linear is fine. That’s ten coefficients.

Then you go to quadratic. How many coefficients you got? Come on, a quadratic and ten variables, how many coefficients? I always have two answers, one of which is wrong, but not – it’s only off by a factor of two. It’s about 50, right? You have 50 coefficients in it, and then you go, “Well, how about cubic?”

Well, then you get 1,000 divided by three. You get 160 or something like that in a cubic. And so the problem is if you have limited data, when you do polynomials, if you do generic polynomial fitting, you just get piles and piles of coefficients. Everybody see what I’m saying here?

Okay, so this sort of changes that because you could do these methods with sparcity. You can fit a polynomial. It will be a nice sparse polynomial. It might turn out that some sick polynomial degree four that this thing picks out that works really well. So okay, I think I said enough about that.

Sparse signal reconstruction, this comes up, I guess as people doing it. There we go. There’s one. There’s another one though. There, there you go. There’s two at least. Okay.

So sparse signal reconstruction goes like this. It says estimated signal. You’re given some noisy measurement, and you’re given the information that X is sparse, okay. That’s like prior information you’re given. That’s – so that’s what you’re given.

And that comes up in a bunch of cases, like it could be false anomalies, which I guess it is in your case. What is your – oh, it’s sparse coefficients in the radon transformer. It’s gonna come up a lot. Okay, and the – if you want to do maximum likely estimate, and I tell you that something has only K nonzeros, you would end up minimizing the subject to that. I mean you can’t solve this problem, so but you’re gonna do an approximation of it.

So and this is actually very important. You should think about this, about sparse [inaudible]. It’s very, very interesting to think about, and I’ll give you an example. If you go back to sort of the last – if you go back to sort of the last century, everything is based on kind of lease squares. And if someone – if there’s a statistician who says, “Justify that,” you would say Gaucian or something, and then that would turn into some now statistically defensible reason why you’re using lease squares, right? That’s how these things work.

But sparcity doesn’t really come up, so sparcity would come up and once you start thinking about it, you start realizing, well, there actually are lots of cases where a single model is that it’s sparse. One would be like set commands or something, set point commands or something like that. If you build some sort of a controller for this based on all the standard stuff, you’re really saying that the input is some Gaucian process.

But in fact, what you really might think or want to say is something like this. No, the input actually is constant for long periods of time and then switches to another value. That’s actually what – that’s the way set points and commands actually look. That’s sparcity, so that would be that kind of thing.

Okay, so we’ll – okay. Estimation with outliers is another beautiful example, and it works like this. You have YI is AO transposed X plus V. This is just a linear one to make it simple. The VI’s are our friends that were with us last century. Those are our nice Gaucian noises, our catchall. It’s basically this is our statistical excuse for using lead squares. That’s what this means, okay.

But then you have this other weird noise, W, and the only thing you assume about it is that it’s sparse. So, for example, it has fewer than K entries or something like that. And basically what these are is these represent either anomalies, outliers, or bad measurements, so – and this comes up a lot.

So basically this says that every 100 samples, the measurement you get is just completed unrelated because you don’t assume anything else about W. And, of course, by choice of this thing, I can make this thing anything I like. So it’s something that’s complete – just simply unrelated to the input, but you’re not told that. So you’re not told that. You’re simply told, “I’ll just give you the measurements.”

By the way, if these are all on the order of like – on the order of one and W is like ten to the five, this is not a hard problem because the measurement comes in. They've all been coming in around plus or minus one, and one comes in as ten to the five. It's a good guess that's an outlier. Anybody can handle that, okay. The really insidious, cool way is this, is when the bad measurements come in, they have the same statistics as the good ones.

So when you have a measurement of .8, whatever is plausible, and the completely bad measurement comes around, and it's .8, that's much trickier to do. Actually, you'll find out these methods that we're gonna look at here, they'll handle that like amazingly well, I mean amazingly well where I don't know of any other method really that would sort of get close to doing this.

Okay, anyway, so you get this. You're basically doing – you're estimating the set of outliers and so on, and it would look like a problem like that. By the way, you'll see that when we replace this with an L1 norm later, this is gonna recover robust statistical methods that have been used for 25 years in statistics, so okay.

Here's a very interesting one. Suppose you have a set of convex inequalities, and they're not feasible. They're not mutually feasible. So, for example, let's do this. Let's make these a bunch of design requirements. Oh, let's make it a circuit. So this is a power limit, an area limit, a timing constraint or something like that. They're not mutually feasible. They cannot be satisfied, so the intersection is empty. Then you'd say this.

You'd say, "Well, first of all, that's good to know," because you'd go back to the person and say, "Your specifications are infeasible." And they'd go, "Well, you can't do it. Maybe I need another designer." And you go, "Oh, no, wrong. This problem is convex. And when I say it's infeasible, I don't mean I failed. I mean everyone will fail to do it." And they go, "Well, uh huh, fine." And then they go, "That doesn't help me. Of the 17 constraints, please come back with a design that violates as few as possible." So that's how you could do it.

And in fact, what you could do is you could classify your constraints into the ones that are non-negotiable. And you could then have the other ones that you really wanted, but can't have. And for that latter group, you say, "Okay, of these 10 or 50." Oh, if I said ten, it's no problem because two to the ten is 1024, and that you can solve immediately, so I should have said, "Of these 100 inequalities here that I'd like to have, please find me a solution that violates as few as possible." Everybody see what I'm saying here?

Okay, all right. So you would do this. You'd say, "Minimize the cardinality of a vector T subject to FI of X is less than T, and T is positive." Now what happens here is this, is whenever T is zero, it means that the Ith inequality is actually satisfied. If you can't satisfy the Ith inequality, T is sort of like a little fudge factor. So T positive means you've been given some slack here. So this thing here will immediately and directly solve the question of finding the minimum number of violations.

By the way, these are really useful. By the way, we'll see also, you've already seen this, you've used it, we've talked about it before. It's an L1 phase. It's a sum of violations phase one method. So, but it's good to see all of these in the same context. Here's one, linear classifier with the fewest errors. So we'll do some machine learning type stuff. So I have a bunch of data. I have X1, Y1, XM, YM. These are vectors, and then these are binary labels, so they're plus minus one.

And what I would like is I would like to find a vector W and an offset V so that sine of W transpose X plus V is Y. Now approximately equal means something like you make as few errors as possible or something like that. You do it right if YI times this thing, actually, if YI times – if the sine of this – well, if you make this positive, that means doing it correctly, if these have opposite signs, that means you've messed up.

So if you want to find the W and V that give the fewest classification errors, that's basically you have a bunch of things that are labeled. If you can get zero classification errors, that means you can put a hyperplane between the points, but now you want to put the hyperplane there that has the fewest number of things on the wrong side, and that would just be this problem here.

Now I should remind you something here, and that is that not one of these problems that we're writing down can be solved, so that's important to understand. They're all hard, so you might ask why are we writing these down, right. Oh, we can solve them in some cases, right. If there's ten – if I've got ten weights, I can solve this, okay. If this is ten features, I can solve this problem by solving 1,024, sorry, yeah. No, sorry, if M is ten, I can solve 1,024 convex problems, and I will get absolutely the linear classifier that has the fewest number of misclassifications. But if M is bigger than that, you can forget it.

We're writing these down this way because later we're gonna see a heuristic that's gonna apply to all of them, and you're gonna get good methods.

Student: Why is there a one?

Instructor (Stephen Boyd):Yeah, why? Because to make it right is to make this positive, to make this thing positive. Yeah, so you can't write down a strict inequality in a problem. Well, you can write it down all you like. It won't do the right thing. You need strict positivity.

However, this – that constraint, when I make his strictly positive here, is homogeneous scaling WNV, and therefore, to say that it is positive is exactly equivalent to saying that it's bigger than or equal to one by scaling WNV. So this is a standard trick for that. Homogeneous problem with a strict inequality, you kind of replace it with a one.

Okay, here's a variation that's actually extremely interesting. This one is a variation on finding a point that violates as few as possible. In fact, it's the dual, and it works like this. I give you a set of inequalities, which are mutually infeasible, and what I want to do is the following. If someone says – if you said, “Well, I have 28 specs, let’s say on phase, margin, power, rise time, slough rate, blah, blah, blah.”

You’d say, “I have 28 of them. They’re mutually infeasible. No one can design that amplifier period.” So then you say, “Okay.” The question then is what’s messing you up. So and let me – first, let’s figure out what would be a powerful statement. A powerful statement, if I gave you 28 inequalities, would be to say, “Well, actually, as it happens, inequality number 14 alone is infeasible.”

That’s a very powerful statement, right, because it means that basically you have to deal with that one. If you’re gonna relax one, you have to relax that one. Everybody see what I’m saying? A slightly less powerful statement would be, “Well, 14, 22, and 37, those three together are mutually infeasible.”

And the weakest statement I could make is that they’re all mutually infeasible, but if I remove any one inequality, they become feasible again. Everybody see what I’m saying here? That’s the weakest statement you can make. So basically the smaller the cardinality of a subset of infeasible inequalities, the stronger the statement, the better you have identified what the problem is, what’s causing the infeasibility, okay.

So how do you do that? Well, you do it as follows. You look at the dual function. That’s G of lambda. That’s this thing. That’s again by homogeneity. It has to be bigger than one and lambda positive. And so what you’d do is you’d minimize the cardinality of lambda subject to G of lambda bigger than one, and lambda bigger than or equal to zero here.

If lambda is zero, it means that you – if lambda I is zero, it means that you didn’t use FI in constructing your proof of infeasibility. And it means – so a sparse dual certificate basically corresponds to identifying a small subset of mutually infeasible constraints, so that’s how that works.

Okay, we’ll do another one. It is portfolio investment with linear and fixed costs, so let’s see how that works. We’re gonna – we’ll take long positions, so we’re gonna invest an amount, XI, and they’re gonna sum up to be here, but we’ll get to that in a minute. No, that’s our budget. Sorry, they’re not gonna come up to B because we’re gonna have to buy.

We’re gonna actually gonna have to – we’re gonna purchase these things, and the way that’s gonna work is this. So here I’m gonna have a trading fee, which is sort of alpha. This could just be a scalar. It doesn’t really – well, sorry, a scalar times one. If I buy an amount, I’ll charge you something like 1 percent, but also if you buy a stock at all, if you initiate a trade, I’ll charge you $50 bucks plus 1 percent. That would be for like a sad consumer like us, or something like or something like that, right?

So that’s the way this would work. So the budget constraints is this. Let’s see. That’s the total amount of the stock that you – that’s the value of the stock you purchase, that’s your trading cost, and that’s less than your budget. That’s it. And you just say, “We’ll just do a simple 1952 mean variance problem." So your mean return on the investment is miu transpose X, and the variance is X transpose sigma X. And so you might say something like this. "I want a minimum return of R min, and I would like you to minimize the variance, the risk of this portfolio, okay."

And then this is the constraint that says that you're sort of within budget. Now, of course, when you do this, this will be equals here, okay. So that's a relaxation. It doesn't matter because you can't solve that because of this thing here. Now this one is actually quite interesting.

What it says is well, let's even just talk about it intuitively for a bit. If you invested in one stock, then your fixed trading fee is gonna be very small. It's just gonna be beta. So you're only gonna have lost, let's say, $50 bucks if you invest in one. Now the problem with that is you may – you're gonna get high risk. The whole point investing in a couple of them is to have the risk go down.

So if you have a couple of assets with a correlation coefficient that's away from one, I mean ideally it would be negative, in which case you can hedge, but that wouldn't actually happen. But let's suppose they were a reasonable correlation coefficient was not .9 or .8. If you invest in two, your risk goes down. The problem is if you invest in two, you're paying $100.00 in fixed fees.

So this is kind of the idea. If you solve it without – if you ignore – if you say beta equals to zero, that's a convex problem, and you can actually find out. That'll generally, by the way, invest in a bunch of different things, okay.

Let's do piecewise constant pitting. So here you have a fit, you want to fit a corrupted signal, X corrupted, by piecewise constant signal, X hat with K or fewer jumps. So in other words, I have some measurements of – I have a signal, which I believe to be piecewise constant and to make a jump every so often. And over my time of interest, I believe it's gonna make K jumps, but no more, okay.

I can give you very noisy samples of it, and you want to fit it with a signal that's piecewise constant. All you have to do now is this. You form a difference matrix, and DX hat is actually the difference vector, and the cardinality of that is the number of jumps. So if it's N minus one, it means that at every step, X hat changes. If it's zero, it means that X hat is constant, it's flat.

Okay, so your problem might be this. Get as close as you can to the corrupted signal with a piecewise constant signal that has K jumps. By the way, some of these you could actually solve. This one you could probably actually solve. That one's probably not hard.

You could also do piecewise linear fitting. So piecewise linear fitting, you do the same thing, but you take a double difference operator, so a del square operator or something here. So you take the second difference because this thing gives you the second difference. And a jump, if del X hat is zero, it means that X hat is the average of its two neighbors, and that means there's no curvature there. There's no curvature. If this is sparse, it says that the signal X has piecewise linear. So this is a piecewise linear signal with K jumps, so okay.

Finally, we'll get to the L1 norm heuristic. It's embarrassingly simple. The simplest version of it, we'll look at much more sophisticated ones later, but the embarrassing one is this. You simply replace the cardinality with some multiple of the one norm. It's that simple, or you add it to the objective, and then gamma is a parameter that you use to tune things.

Now we're also gonna see weighted ones like this. But actually, before we go on to that, I want to just draw the picture over here just to show you how completely ridiculous this is. So here it is. Here's the actual function. Here's the cardinality function. I'll just draw it like that so you can get the idea. There's the cardinality function.

And this heuristic says, "Approximate this function with an absolute value." So I don't think there's no one. There are very few people that would argue that this is a good approximation of that. I mean that takes some serious storytelling abilities to convince someone that that's a good approximation of that. And, in fact, when you look at this picture, because that's all you're doing. You're placing a function that looks like this.

It's one everywhere. It drops to zero right at zero. You're replacing that with an absolute value. So the whole thing looks highly suspect, highly implausible, doesn't make any sense, but that's it. Remember the same thing goes with La Grange duality. I don't know if you remember that. You approximate the – your irritation function that looks like this with slope, so – and that worked out. In here, it's not gonna work out. It'll be a heuristic.

Okay, so let's just do an example here. We start with this hard problem. Find me the sparsest X that's in some convex set. The heuristic says minimize the one norm of X subject to X in the convex set. This is convex. That's an easy problem to solve, and what can you say about it?

The solution of this is not the solution of that. We'll see what you can say, there's cases where you can get a bound on this solution from that, and there's a very, very small number of cases, extremely specially. Basically C is an alpine set, so it's just a quality constraint in which there's actually a statement you can make.

However – I mean a statement that's actually true. You can always say this. This seems to work unusually well as a heuristic, this and variations. So as a cardinality constraint problem you could do many things. You could add this this way and then sweep beta. You could also add it in here. And you would adjust these numbers so that the cardinality is less than K.

And I think I'll mention one quick thing. I think we'll do one example and then we'll quit. So polishing is quite interesting, and let me explain what that is. It's not universally used. Statisticians don't do it for some reason, and they tried to explain it to me and I didn't get it. Apparently it's bad, but anyway, at least in the context of their problems. But for other applications, it's not.

It works like this. You use the L1 heuristic to find an X hat with required sparcity. Now once you've done that, you fix the sparcity pattern, you go back and you solve the problem again. That's how that works, and that's called polishing. That's not a standard word. I just made it up, but it sounds like a good word, so a good description of it. Are you doing that?

Student: Not yet.

Instructor (Stephen Boyd):You will now.

Student: Yeah.

Instructor (Stephen Boyd):Okay, good, so good. Yeah, it doesn't take much, okay, so to do this. And I think what we'll do is maybe we'll quit here. Actually, what I would like to do is just to get to a quick example, and then we'll come back just for fun. There we go, okay. So we're gonna do a quick example and then just so I showed you one.

So we want to find – this is the regression of selection problem. We want to select columns of A to fit B with, K of them. And so what we'll do is we'll replace this. We'll do an L1 thing. You actually – you add this plus lambda times the one norm of X, and you trade it off. This is a tiny little problem, although it's already big. I could actually do this if I wanted to because two to the 20 is about a million.

And so this one, I can actually find the global solution. In fact, I believe this is the global solution. This is the global solution calculated with a million – by solving a million lead squares problems. What this shows, the solid curve is what happens if you do the L1 heuristic with polishing, and you can see it works pretty well. Everyone agrees sort of here, but what happens now is – and you can see any gap between these two is where you've missed the global optimum.

What's interesting is, for example, over here with three, you actually get the global optimum. In other places over here, one, you get the global optimum. I guess you nailed it at six as well, but notice that you're never really very far off. So and considering that the effort in calculating this curve and this curve in this case is off by – is a factor of ten to the six. You'd have to say it's not bad. I mean this is not bad. So for solving at a million times faster, you get close.

So okay, so this is a good time to stop, and then we'll continue next time. And oh, before you leave, I just – there is a tape ahead tomorrow, 12:50 to 2:05. It's on the course website. We'll quit here.

[End of Audio]

Duration: 78 minutes