

ConvexOptimizationII-Lecture12

Instructor (Stephen Boyd): The first is your revised proposals are due tomorrow. So – and we would actually like those maybe stapled – not stapled to – or maybe paper clipped to the original one, so we can see what the original one looked like and so on. As I've said a couple of times, you're more than welcome to grab any of us, me or the TAs, between now and then to sort of just glance over what you have or -something like that. The way you know it's right is that you can explain it to somebody in way under one minute. So if you can explain it in under a minute to another person, not working with you on the project, then that's a sign that you can articulate it completely clearly and all that sort of stuff. Okay. So those are due tomorrow.

And then the other thing is that a week after that will be these midterm progress reports. And then that's basically the same thing. It's – takes your three page thing, but now it's four pages, or maybe even five. So – because it's now got – this one should be your statement of what you're looking at, what the real – you know, what the problem is, and maybe something about the approach. That's what we should be seeing tomorrow. And by the week after, especially because these days it's not a huge deal to actually implement any of these things. It takes, like, 15 minutes to write scripts for a lot of these things. Then by next Friday, we'd expect to see at least some results about how these things – I know that some people are way ahead on that. And there's actually two different – some people are way ahead on that, and maybe way behind on the writing, and vice-versa, and stuff like that. So we'll make sure to keep those sort of on track. Oh, the other thing is I guess we're confirming May 30, which is the Friday – the week before the last week of classes – as the due date for the project reports. So – of course, by then we'll probably have read your project reports about 45 times. So it'll maybe just be a formality at that point, which is fine. We don't mind doing it. So as long as we see serious progress each time we take a look at it. So that's gonna – that'll be that Friday. And I think what we're gonna do is we're actually gonna have a – there'll be poster session, and it's gonna be maybe an afternoon or evening, the week of the last week of classes. So that's, like, June 2 or something like that. But that's – it's on the website. And so that's what we're gonna do, and it'll maybe be some evening or afternoon. And the posters – I mean you'll hear more about this, but it's totally obvious what it is. It's your standard 12 pages and 12 slides. Nothing fancy. This will just be – you'll print out 12 pieces of paper. We will have to figure out where on earth we're gonna get a bunch of poster boards or something like that, but we'll – seems like we can figure that out. So we'll do that. And it'll be interesting. That'll be for everybody to see everybody else's project. If a project is well – if it's already – if it's well written, it's shockingly easy to write a talk. So – in fact, you'll find out you've already written a talk because the sections of your report, or whatever it is – I guess it's the report at the time, will simply be a talk, except it will be in 12 point, and you just blow it up to the bigger one. You can use our – we have some templates for talks. So.

Let's see, a couple of other things. Oh, you should watch for email, and maybe the web site, because we may tape ahead next -Tuesday's lecture tomorrow some time. So – and we would. You'll hear about that. Okay. So, I can't think of any other administrative

things. I guess we posted a new homework, but I guess most of you know that. Any questions about last time? Because if not, what we're gonna do is we're gonna finish up this topic of Heuristics based on convex optimization for solving the non-convex problem. So that's the first thing – we're just gonna finish that up, and then we'll move on to another little coherent section of the course, which is basically how to solve absolutely huge problems. So for some of you – I mean if you've already chosen a field, and it's too late, this will be relevant. –And if you make the mistake of -choosing a field where you have to solve huge problems then this will be – these will be the – it would be two or three lectures – these will be for you. For those of you who haven't chosen a field yet, this will give you – it'll let you – this will be encouragement to not choose a field where vast and huge problems are requires. So that's my – that will be my recommendation. Okay. So let's do difference of convex programming. Let's finish up this step. I think we looked at this last time. So this is a topic or method where you write all the objective and inequality constraint functions. You can do with equality constraints functions, too, but I'm not gonna go into it. And what you do is you write them as differences of convex functions. All right, so convex plus a concave function. And then the obvious linearization – convexification, I should say, is this. You linearize the concave functions. So only the concave portion – that's minus a convex one – is – you linearize that. And then this is roughly speaking, based on the idea – I mean you can make this precise, but it doesn't need to be – is the that the best affine approximation of a – the best convex approximation of an affine function is an affine function. All right. So – okay, so you do this and the result – this is now convex because the convex part you left exactly as it was, and the only thing you've done that's a constant is linearized – that's this thing – the concave portion. So – by the way, in a – this – the way to do this is, for example, in – it's actually pretty straightforward. –The way you would do this for a quadratic, you can work out how that is. If you have a quadratic that's neither convex nor concave, that's split eigenvalues. You split into the positive part, and that's gonna be this thing. And then the negative part, you simply linearize, but you drop – you will drop the quadratic term. And then that gets linearizing, it's just dropping the quadratic term. That's only for the concave part. And actually, you'll find out it's quite interesting. If you have a function that's convex, that this -difference of convex, concave, it'll actually have some negative curvature. And basically, what you're doing is your just flattening the negative, but you've retained, exactly, all the positive curvature -portions. Okay. So an example we'll look at is from the book. It's estimation of a covariance matrix from samples. So you form the sample mean, and the negative log likelihood function is $\log \det \Sigma + \text{trace}(\Sigma^{-1} S)$.

Now, I mentioned this last time that the general approach now is to do the following: is not to estimate Σ , but Σ^{-1} , which is – that's a one-to-one transformation, so you're perfectly welcome to do that. In which case, this function is convex in Σ^{-1} because that's linear, and that's convex because it's logged at Σ^{-1} . Well, it's logged at Σ^{-1} inverse, so – which is convex. So that's the standard method. Now, that works, provided the constraints you want to impose on Σ are convex in the inverse of the function – sorry, inverse of the covariance matrix. Actually, there's a lot of interesting constraints that are convex in the inverse of a covariance matrix – many. But some are not, and actually, we just want to illustrate DCC, or whatever this is, a convex,

concave procedure, something like that. So we just picked on that's not. So suppose I told you that all the variables are non-negatively correlated. There are no negative correlations. So that would – that's this. And this is not preserved under inversion. So the set of matrices whose inverses are non-negative is not convex. Okay. So that leads us to this problem here, and we have a convex part and we have a concave part. And we linearize the – we're gonna linearize the concave part. That's this. I mean this doesn't matter because it's a constant. And we're gonna preserve this convex part. So – and the results here, I think we looked at this last time. I just wanted to make sure I went over this – well, is this. And my guess is that this is actually globally optimal. And the reason is actually there's – well, there's even an exercise in the book that goes over this. It turns out that in some cases, this problem is actually convex insigma, depending on the final estimated sigma versus the covariance matrix. So that's this. It's just to show you how it works. Okay. We'll look at alternating convex optimization. That's our last topic. And we'll look at a couple of examples. So this is another idea that comes up. Lots of times, it's discovered periodically, will be discovered periodically in the future by many people in many different – maybe one of you will discover this again sometime in ten years, or even less. That's fine. I mean, actually, there's nothing wrong with discovering it. The only thing wrong with discovering it is imagining that, in fact, you're the first to discover it. I mean that's the only actual problem here, which there are people who do. So, okay, all right, so this works like this. You partition – these aren't actually started – not partitions. I have index subsets here. And you assume that for each of these index subsets, the problem is convex in a subset of variables.

So that's sort of the – that's the picture. Okay? And in fact, we can -even, if you like, look at an example just for fun because that's the only way this really makes any sense. I wonder if you can raise this screen up to about here. That'd be great. Okay. That's good, -right there. So let's take a look at that, and actually, let's look at something extremely simple. Things don't come simpler than that. And let's imagine the variables are actually, for some reason, A, X, and B. So that's three groups of variables. And now, I want to know is that convex in A? What I mean by that, of course, is that X and B are fixed. Is that convex in A? Of course it is. It's affine. It's a norm of an affine function. So it's convex in A. Is it convex in X? Obviously. Is it convex in B? Of course it is. Okay, now, let's do pairs. Is it convex in X and B?

Student: Yes.

Instructor (Stephen Boyd): Yeah. Is it convex in A and B? Obviously, right? And the final one, is it convex in A and X? And the answer's no. Okay? So this would be a case where there's all sort of index sets. So the index might be something like this. And you might as well get maximal ones, right? So AB – I guess that's about it, right? If you're gonna do that. And you could have XB. Maybe these are the maximal – sets, something like that. Those are the maximal sets, index sets over which the problem is convex in each set. Okay. Now, the method now goes like this. I mean it's really kind of dumb. For each of these index sets, you – it's convex optimization to solve that problem. So you do that, and then you kind of go back and forth and so on and so forth. And hope that this works. So actually, let's even look at an example. Suppose I ask you to solve this

problem. So here would be the problem. I should remember those sets, actually, AB and, I think XB . So, in this case, we're asked to minimize this – note my period – okay, subject to that X in some – X – this is convex. Right. A and some script A and B and some script B , okay? These are all convex sets here. So obviously a non-convex problem, that's clear. It's non-convex. Why? Because of the AX there; actually lots of problems come up this why, incidentally. In fact, there's even projects that are literally, directly this blindly convolution. I think – is that yours? Yeah. It's yours. So there's plenty of things that come up looking like this. And so this method would look like this. It would alternate, and it would actually say first, fix X to the current value. And optimize over A and B . And that's a convex optimization problem. That would update A and B . Then it says fix A and B , and optimize over X and B . So notice that B is being optimized, actually, every step. I mean this is not a big deal, but it's just to illustrate kind of how this works. So that's the picture. Let's see. What can you say about this? Nothing. I mean that holds for this entire lecture.

But, I don't know. You can – I mean you can say trivial, stupid things. But after a while they get kind of – you know, you realize, like, why bother? I mean, you know, if there's theorists – I don't know. It's – look, obviously when you do this, every time you do it, the objective goes down. So amazing, isn't it? The objective actually – it's a number that goes down. Therefore, it converges – I mean, if it's bounded below. So – I don't know. And then, so what? So – okay. I mean I don't know, that's better than nothing, or whatever. But it's so obvious that I don't even see the point of stating it. Okay. So a special case is actually when you partition the problem into two variables. And you just alternate. But there is lots of variations on this. That's the nice part is because it's an algorithm that often works, but need not. It means, actually, that for this whole lecture on non-convex optimization is that there's lots of room for personal expression, which is not the case, basically, for convex optimization because, you know, a method that solves a convex problem, like – doesn't badly is just called a non-method, right? And any two methods get – I mean if they disagree on the answer, one of them is – I mean and one of them is a not acceptable method, right? Well, for a non-convex optimization, who knows? It's very nice, you know? Maybe your method works better one problem instance; mine on another. I mean, so it's very nice. You get room for personal expression. I can mention some of the variations that you can do for this. One is actually to not optimize all the way, but to take a fractional step in between. So that's an option. It doesn't sound like it makes sense, but actually that can actually make these work better and so on and so forth. But just remember this is – that's alchemy. That's just like – who – it's just go ahead and try it and see what happens on your problem. So that would be – in other words, you would optimize, for example, over A and B , and not step A and B all the way there. But step them half – some fraction – halfway. Okay. All right. So this is alternating convex optimization. Here's a recent and interesting application of this is non-negative matrix factorization. If you don't know about this, you probably should. Although the people popularizing it are making way too big a deal out of it. But that's okay; they're welcome to do that because it is really cool. Here it is. It's sort of like – you might call it a signed, singular value decomposition. So basically, I give you a matrix. I want to explain it, somehow, as a low rank matrix, as an outer product, something like this. And you should imagine that K is small, here. So K – you might want to approximate matrix A as an outer

product of a – I guess in this case, a M by five, and then five by M . So you'd have sort of, like, five factors or something.

Now, without this, this is completely solved. It's just the SVD. You take the SVD and you truncate the SVD expansion, and you get the answer. Let's see – oh, by the way, it's EE even – I mean there's lots of – it's not unique here, and that's completely obvious because I could – I can put a positive matrix – you know, K by K inside here, and I get the same – I get a different factorization that yields exactly the same objective value. So it's not unique. But that doesn't really matter. Let's see, a couple of cases we can solve. Oh, you do K equals one. So we can – you can actually approximate a matrix in Frobenius norm, well, any norm. It doesn't matter, a Frobenius norm or a spectral norm in – with a outer product, which is restricted to be positive. And that goes in two steps. You first take the positive part of A . That's the first thing you do. And the next thing you do is you take the singular value decomposition of that. Now, it turns out there's something called Perron-Frobenius Theory that says that the singular values of a positive – singular vectors of a positive matrix, the first ones associated with sigma one, which is what would come up in the rank one approximation, those can be chosen to be positive. Okay? So that would – that's one case where you can do it.

Okay, now on the other hand – I mean come on. It's just this thing. You look at this and you realize, "Wow. It's certainly convex. And X of Y 's fixed. And it's convex in Y if X is fixed." And so you can simply alternate over these. And you can say a few things. I think in one of these variables, it's actually separable. I forget which one. I don't think it matters much. Maybe in the columns of Y or – I – it doesn't matter. It's separable. Actually, the separable is not that exciting unless you're gonna do – unless you're gonna run on many machines. Unless you're gonna do an MPI or something like that. Because, as we've discussed several times, if a problem is separable, it just means the computation time grows linearly with the problem size. But that's the case even if there's light coupling by these sparse methods. So, okay, this is non-negative matrix factorization. And here is an example with a 50 by 50 matrix. And we want to show it – we want to approximate it as a rank five matrix. And then, here it is starting from, say, five starting points. And you can see it's going down like this, and will converge. It need not converge to the – in fact it's very – it's unlikely to converge even to the same – it might converge, in this case, in the same one, but there's no guarantee and all that sort of stuff. So all of these methods – these non-convex methods, they're extremely useful. So the fact that no one – you know, if you're doing the Netflix challenge or something like that, and this method works, I don't think you're gonna sit around and complain and lose sleep over the fact that you're using a method that you can't prove always gets the global minimum. So this lots – you have to remember there's lots of cases where this stuff is fun. It is actually – I mean it works perfectly well. If you don't get the optimal trajectory for your robot, as long as you reliably get a really good one, that's fine. There's other cases where you can be much less – you can't be as casual about not finding the global solution for something. But this one – yeah?

Student:In this problem, did you create A to be the product of two positive matrices?
[Crosstalk]

Instructor (Stephen Boyd):No, we didn't. I think it's just random. All the source codes are online. You can look at it.

Student:If you did this, would you get the answer, usually? Or –

Instructor (Stephen Boyd):I think you said it just right. Yeah. So yeah, the obvious thing to do here is to make $A = XY$ – you know, generate A as XY and see if you get back XY . By the way, you won't get XY , right? Because there's a – what do you call it? It's not unique. But you'll get something – you should get – they should have the same ranges or whatever. They should be unique Modula of this transformation. Sorry, you should get the right X and Y back. Yeah. I think what happens in that case is you often get back. So I think you said it exactly right. You usually get back what you were looking for. So it's heuristic. You do not have to get it back. For one thing, it depends on where you start from. So – yeah, but these things work, like, pretty well, often. Okay. So that finishes up this. I should say that this was sort of – it was out of order for the class. It was out of order because it – that was just a whole lecture on non-convex methods kind of stuck in the middle of the class. It kind of logically goes at the end, but I thought we'd switch it around for the benefit of those doing projects that are non – involve non-convex problems. Yeah?

Student:[Inaudible] SVD solves like a non-convex problem.

Instructor (Stephen Boyd):It does. That's right.

Student:So are there extensions of SVD that are guaranteed to find the [inaudible] –

Instructor (Stephen Boyd):Yes.

Student:– somewhat similar non-convex problems.

Instructor (Stephen Boyd):Good. That's a great question. So a good question is, "What are the problems that aren't convex that you can actually solve?" And now we're back, by the way, now that that's lecture's over, we're back solve now, once again, the – we've just went over the right bracket and the scope. Solve now means solve globally. It doesn't mean this kind of like maybe, possibly get a local solution sometimes, which is what it meant in the last lecture. So we're back to the global definition because that block just finished. And now let's answer your question. So the question is – so clearly there are non-convex problems we can solve. One is things involving like, what's the best rank for the approximation of this matrix? That's – we do that by S – truncated SVD. It's – that is plenty non-convex. And it actually – so the question is what are the others? And there's a handful of others. So there's some – we can solve any problem that involves two quadratics, convex or not. That's just – it's actually not obvious. And then you can involve some weird things, like things involving – some things involving quartacs. But it depends on the dimensions. And these are just by sort of accidents of algebraic geometry. But we're not gonna talk about it. But the point is – I guess my opinion is there's just a handful of problems that are non-convex that you can actually solve. And I mean in the

strong form. Solve as in get the solution. Right? So if that's what you mean by solve, there's just a handful. SVD is one, and you're gonna find plenty of others. But –

Student: Can I ask you a question about quadratics?

Instructor (Stephen Boyd): Yeah.

Student: So if you have a convex, concave problem –

Instructor (Stephen Boyd): Um hm.

Student: Why can't you maybe, like, approximate one of them with one quadratic, approximate the other one with another quadratic, and then inherently use that to solve –

Instructor (Stephen Boyd): That doesn't solve the problem. What you can do is you can minimize any quadratic function, convex or not, subject to any single quadratic inequality, convex or not. Okay? You – by the way, you know some examples of that, like, how do you minimize $X^T A X$ subject to $X^T X = 1$? Answer? Minimum eigen vector. Okay, and that's a non-convex problem, if A is not positive semi-definite. If A is positive semi-definite, well – well, actually with the constraint there, $X^T X = 1$, it's non-convex there. So yeah. So there's a handful of these, but not a whole lot. But they're good to know about. I mean, so – that's actually in the appendix of the book, if you want to read that. It's an advanced topic, and that also is something, which is – that's actually something deep, unlike difference of convex programming, which is basically just an obvious kind of trick. That's actually deep, and you can actually say – I mean that's a strong statement. And yeah, people periodically – about once every ten years – rediscover that in different fields. It's got – and so it's got lots of different names like Parrot's Theorem – I don't – you – look in the appendix because we listed some of the things there. Okay But if you have two quadratics, you can't do it. Although there are some weird, isolated cases where you can solve it with two quadratic constraints. So this is – it's weird. And then there's weird things where there's, like, quartacs. And so there's just a handful of random things that people who study these things would know. And by the way, in almost all of those cases, in a – not in the eigenvalue case, but in some of these cases, it turns out that these non-convex problems you can solve are basically because you form a convex relaxation, solve that. And then you prove separately that the relaxation actually can always be rounded to – it's always tied or something like that. So that's how those work. And those always involve real stuff. If you look in that appendix, it's not simple. So – okay, so that finishes up that. Like I said, we had a right bracket, so that means we're back to convex optimization again. And we're gonna start another chunk of the class. It's another coherent chunk of the class. And it's basically on how to solve extremely large problems. So basically, as big as you want, if you can store the data. I mean basically, we can solve the problem with a memory that's basically on the order of the size of the data, number one. And with a computation that's completely linear. Right? So when you get to problems that are a million variables, ten million and more, N^2 doesn't do the trick, basically, either in storage or anything else. So – I mean not even remotely close to

that. You can go – you can handle an N to the one point one, or something like that. Or one point two. Beyond that, forget it. So we're gonna look at these methods. They're actually extremely good and extremely cool. And I think just any educated person should know these things anyway, even if you've made the wise choice to go into a field where the number of variables is and will remain for the foreseeable future, under a thousand or ten thousand. But if you're in a field where things are gonna grow, this is gonna be part of your future. So, okay. So [inaudible] radiant method – by the way, how many people have seen this somewhere? Oh good. Okay. Where? Some ICMA thing? Or – okay, so you'll know everything.

Student:CS 205.

Instructor (Stephen Boyd):CS 205. Oh, okay. Great. That was –

Student:Mathematical methods for Vision, Robotics, and [inaudible].

Instructor (Stephen Boyd):Oh, okay. Good. Okay, good. Great. So a bunch of you have seen it. You probably have seen more of it than I have. But we'll – so for you it'll – you can just sit back and, I don't know, relax or something. So for the rest of you, it's great stuff to see. So first, let me start with just sort of the really big picture about solving linear equations. Now, I'm sure all of you know the following. I've said it enough time, but it's important enough. I'll say it again. If you profile, most – certainly, most convex optimization methods, if you profile them and ask if it's – while it's running, what is it doing? It's solving linear equations. That's it. And there's a bunch of other stuff, but it just doesn't even show up in profiling, like, you know, the little line searches and evaluating logs. Zero. What really matters is computing the steps. And that's solving linear equations. So, okay. So – by the way, it's true for a lot of problems. It's just solving linear equations. So let's talk about, at the very big picture, how do you solve $AX = B$? Okay. So the – it's kind of a crude classification. It's actually wrong because there's – the boundaries between them are gray. But, in fact, it's helpful to classify these into three huge classes. And here they are. One is dense direct. These are factor-solved methods. We covered this in 364 A. You've probably seen it other places. This – I mean the software involved there is almost universally BLAS in LAPACK. That's – I mean if you use Matlab, that's what you're using. Actually if you use anything reasonable, that's what you're using. So these are factor-solved methods. You do – you factorize A into a product of matrices, each of which is easily solved. And then you do a back and forward substitution or something like that. Now, the – here, the high-level attributes of these methods. You can take an entire class on just this, right? No problem. It's like CS 237 or something like that. But – and now we're going through all the horrible details of how do you do the factorizations? What are the round off error implications and all that kind of stuff? But at the highest level, here's what you need to know about these methods. The run time actually depends only on size. That's not completely true, except for positive definite systems, in which case it is true. So for a indefinite systems, you do dynamic pivoting, which means that you actually – there actually are conditionals in the code. You look and you look for a new thing, and you reallocate – and unless you've allocated

memory correctly ahead of time and stuff – you know, if you – well, no, I guess in this case that wouldn't happen. Sorry, scratch that. That's coming up.

So there are conditionals, and that means that actually solving $AX = B$ for a general matrix A actually is not independent of the data. If I give you one A or another A , it can actually take slightly different amounts of time. Actually, it's absolutely second order, so you won't notice it. But it's not deterministic. If A 's is positive definite, it is absolutely deterministic because in Cholesky factorization – you do not pivot in Cholesky factorization. And actually a compiler will unroll the loops, and it's just – you know, it's whatever it is, N^3 over three operations, all pipeline, and it just shoots right through. And there's no – it can never be different. It will be repeatable down to the microsecond or lower. Okay? So that's it. Well, there's a few other that happen. Of course, it can fail. I mean there – it can throw an exception if you pass it a singular A . So – but roughly speaking, other than that, it's just – it's deterministic. It has nothing to do with the data, except of course when to the boundary where A is singular, it doesn't have anything to do with the structure of A . If you toss in a tridiagonal A or a sparse A or banded A , this thing doesn't care. Okay. And this will work well for N up to a few thousand, I mean immediately. And you can do the arithmetic to find out. It has to do with how much will fit into your RAM and stuff like that. So this will work up to a couple thousand. If you want to use a bunch of your friends' machines and use MPI or something, you can probably go bigger than that. But that's – you know, you can just – I mean it will actually – this will scale up to much bigger problems, but not on a laptop or a little machine or something like that. Okay. So the advantage of these is that it's very nice. It's just technology. This is just $A \backslash B$ at the highest level, just done. Right? Extremely reliable, you know. When it fails, it fails generally because you gave it a stupid problem to solve. That is to say A was near singular for some practical purpose.

Okay. Now, this is – so this is the first level. The next huge group is called sparse direct. So these work like – direct means you still are factoring A , but this time you're taking into account sparsity in A . And these – I mean sometimes there are well known name structures, like A could be banded. That's a case where it's named. Or it could be a sparsity – just a sparsity pattern in the sense that there's just lots and lots of zeros all around and so on. Okay? So these are – actually those are sort of different cases, but they're – I would call them both sparse. I'd put them in this part. Now here, the run time is going to depend on the size. Of course, that's always the case, the sparsity pattern. And it's almost independent of the data. It's actually a little bit less independent of the data than it is here. And the reason is in the general, unless it's a Cholesky factorization, you're actually doing dynamic pivoting. And in this case, dynamic pivoting has fill-in effects. So you may even end up – you know, if you end up calling `malloc` or something in the middle of one of these things, you're gonna be sorry. So, in fact, the way these typically work is you'll actually allocate – before you even go in, you'll allocate a block of memory hopefully big enough to contain all the fill-in that is gonna happen because as your dynamically going through it, you're generating fill-in because you didn't like the pivot that you would have taken, or something like that. Okay? So that's what this is. And these will – it depends on the sparsity pattern. This'll take you up ten to the – I mean generally, ten to the four pretty easily. Ten to the five if it's special, like if it's banded or block

banded or has some arrow structure or something like that, you can easily take this up to a million. I mean if it's – if the sparsity pattern is – well, let's say it's banded. You can go to a million like that. So my laptop will easily go up to a million. No problem, none. You can just work out. You just figure out how much DRAM it takes. It's just totally straightforward. By the way, this is described, at least in a very high level – more than I'm gonna say about it here. This is in the appendix of the book on convex optimization. Of course, there's entire books on these subjects, too. Many – and a lot of them are really good. So it depends if you want that much detail. But it's all there.

Now, the interesting thing about these is I – you can argue that these are completely non-heuristic. They always work. You know, if you type in a random matrix, they'll dense. And before $A \backslash B$, I'll be able to tell you how long it's gonna take. I mean if I know a little bit about your machine. And I will not be wrong. I absolutely will not be wrong because I know what it is ahead of time. We can guarantee it, all that kind of stuff. I can embed it in a real time code, and we just know how long it's gonna take. End of story. These actually are semi-heuristic. And so you have to be a little bit careful about this. What that means is the following – after a while you get used to these – to sparsity being handled. Then we did – there is some heuristic in the sparsity stuff. And the heuristic comes into the ordering. So how you pick the permutation to do your stuff. That's a heuristic. And there's actually – that's actually a whole field which is very interesting, but there is a heuristic, and it's this. So when you type in your problem, and you have 100,000 variables in $AX = B$. And you type $A \backslash B$, I mean it's as if you're doing mad lab. But remember, they didn't write any of the numerics, right? It's all doing stuff that's – it's written in C and stuff like that, in that case, probably a UMF packer. I'm not quite sure, but that's – I guess that's what it's doing. What will happen is this. You may – there is a moment where you – of tension. And it's either gonna take like two seconds or twelve hours, or something like that. And it's very likely to take two seconds, right? But that just depends on your sparsity patterning and the ordering method you're using. So, okay, okay, and now, we get to the third – oh and by the way, these are not – I'll say it's a little bit about I think the boundaries are gray. Any sparse direct method does this. When it gets near the end, and it's got a little block at the end it's working on, and it's all dense, it – usually when you get down to like, 1,000 by 1,000, it just says screw it, I'm going dense. Even if there's lots of zeros in it, it just goes LAPACK at that point, just BLAS LAPACK. So that's – so a real sparse matrix solver, when it gets – if the fill-in gets big enough it just says done. LA BLAS, LA PACK and does it. So that's why these are – the boundaries not exact here. Now, we get to third class, and it's the one we're gonna study now. It's iterative methods, also called indirect. If you do any – this is basically how all scientific computing does, so large-scale. Your example of that, it's PDE solvers. It's anything large-scale. Typically anything with a million variables and more, you're talking iterative. I mean, unless it's very structured, like banded or something; you're taking iterative.

Now, all of – like I said, all of scientific computing and so on. Now, these methods – and notice that we're losing reliability every time we jump up one. The first one is like it just always works. I don't even have to hear your data, and I can predict exactly how many milliseconds it's gonna work. It is 100 percent reliable, unless you throw in something –

stupid data at it. Then you get to sparse, I can say, "It might – it probably is gonna work. You know, it depends on your data. It depends on the matrices. It also depends on the ability of your – of the heuristic for choosing the ordering, whether that's gonna actually give you a good low fill-in factorization." But after that, it's not – it's probably – after that it's gonna work, once you know the fill-in story. Here, the run time actually depends on the data. So – and the required accuracy. So the other two methods are basically producing very accurate solutions. And it's a real issue, which I'm not covering. But that's – here, this is – these things are tough. These require tuning, preconditioning. That's something we'll talk about. And I'll say a little bit about that. So these are less reliable. They need babysitting. These are not just backslash type things, where – you don't embed these in there. You have to tune it. Now on the other – and you'd say, well, that's a pity. Of course it's a pity. I mean imagine what a pain in the ass it would be if every time you type A backslash B, you actually had to tune or tweak, like, six parameters to make it work. So that's – welcome to the world of super large-scale problems. It's tweaking. You know, that kind of stuff. Preconditioning, we'll get into that. So you could say it many ways. Lots of – there's lots of opportunity personal expression here. A lot of this is not fun, too. And therefore, it often falls to graduate students to do, just to warn you. Just to – I don't – it doesn't mean anything to me, but trust me. This is the kind of thing – ideal for graduate students to do. Okay. And you can complain about this all you like, but the point is, if you're solving a problem with a million or ten million variables, and it's actually – something's actually coming out, you hardly have the right to say, "God. It's a pain in the ass that that poor grad student had to work for three months on the pre-conditioner. And the solutions aren't that accurate." You'd say, "Hey, look. You know, it's your – you were the one who decided to solve a ten million variable problem. Don't – you know, you can't –." There's not much you can say after that. Okay, so this is the third method.

By the way, the boundaries between them are actually very – they're a bit gray. So for example, people will often use indirect – a few steps of an indirect method, or iterative method. Actually, after one of these, that's called iterative refinement. People will also use a pre-conditioner. We'll get to that. Is a pre-conditioner, which is, in fact, based on an incomplete factorization or something like that. And then – so at that point who knows what these things are. So – but it's useful to think of these three huge classes of methods for solving linear equations. That's very useful. Okay, so now we'll get into one, most famous one. Well, they're all – a lot of them are the same. After a while, you get it. So we're gonna look at symmetric positive definite linear systems because actually, mostly that's what we're interested in. We solve either Newton equations, Newton systems, which is $H^{-1}G - H^{-1}G$. Or even if we're solving with equality constraints, we have a big – an H, A transpose A zero, a big KK key system, which is indefinite. But if you do block elimination, you end up solving something that looks like $A, H^{-1}A, \text{transpose}$, or something like that. So a lot of it comes down to solving this. Okay, so this comes up, for example, in – well, in Newton. A Newton step is that. It's just – it's least squares equations, regularized least squares. If you minimize just a convex quadratic function – and of course this is solving – this is also solving an elliptic PDE, like a poisson equation, if you discretize. So that's what you end up doing. So, okay, as another example, it's sort of disanalysis of a resistor circuit. So this comes up also in circuit stuff. So here that you have a conductance matrix, and I is a given source current.

And you want to calculate all the known potentials, or whatever. By the way, same thing comes up in analysis of reversible mark-off chains. That would be another one that I didn't put here, but it would be the same thing. Actually, it's related to this. So if I give you a reversible mark-off chain, or something like that. And I say, "If I start the thing here at this state, and I want to know what's the probability that it hits this state before that one, or something." You're basically solving a problem like that as well. So – or if you want to calculate, like, hitting times and all this, it's the same sort of thing. Okay, so now we get to CG. Actually, there's a bit of confusion over the method. It's sometimes called conjugate gradient, and sometimes conjugate gradients, with the "s" there. And it's kind of – I don't know. I may switch between them.

And then, I think it also depends if your first exposure to this is in a – if it's – I think one is the British style, and one's the non-British style. But I – anyway, but just to warn you, you're gonna see "s" – sometimes "s," sometimes not. Okay, so this is – it was, as far as anyone knows, it was first proposed in 1952, by Hestenes and Stiefel. And it was proposed as a direct method. And now here's the – instead of getting into the details of the method – in fact we won't get to the details of the method until well – actually the details of the method are actually irrelevant. That's probably not how you guys learned it. But we'll get to that later. There's actually many actual methods that will generate the same sequence of points. So – but we'll get to that later. Here's what it does. Here's – high-level attributes are this. It solves $AX = B$, where that's symmetric positive definite. Now, in theory, if you were to do exact arithmetic, it takes N iterations. That's the dimension. Each iteration requires a few inner products in \mathbb{R}^N ; that's fine. And it requires a matrix vector multiply. You have to make it – you have to implement a method that given Z , will compute AZ . So that's what you have to do. That's the – that's basically what you have to do. Okay, now, if you work out that arithmetic of A is dense, then no problem. You call AZ , and if A is dense, you're multiplying N -by- N matrix by vector. That's N squared flops. And congratulations, you run CG. It takes N steps. Each step is order N squared, and you're back to N cubed. Actually, what you have now is an N – is an order N cubed solver that is much less reliable than the standard ones, and actually much less slower because you're using BLAS level one or two. I guess that's BLAS level two because you're using matrix vector multiply, whereas the real method for solving would use BLAS level three and things like that, which would be block, block. And it would use – that would be optimal. Okay, so there's actually no reason to use CG – a CG-type method. I guess, to tell you the truth, CG is really a particular method. And honestly, this really should be called Krumlauf subspace methods. I should even change this slide. I feel so weird because it's really a specific method, but they're all kind of lumped together. Okay, all right, now, if ever matrix vector multiply is cheaper than N squared, you can do better. Okay, so here's an example. If A is sparse, A 's multiplying Z by AZ is fast because you only multiply by the non-zeros.

Now, if it's sparse, and not absolutely huge, you're probably better off using a sparse method, like a sparse direct method, maybe, okay? But let's see, someone give me an example of a matrix, N -by- N , you can multiply a vector by fast – fast means less than N squared – but which is not – but which is full. Not sparse. What's an example? Again, all of you have seen these.

Student:[Inaudible].

Student:A counter-product matrix? Like if it's –

Instructor (Stephen Boyd):Perfect. Actually, that's perfect. Yeah. So a matrix that's, like, diagonal plus low rank. Yeah. Diagonal plus low rank. Oh, I mean you have to know it – represent it that way.

Student:Yeah.

Instructor (Stephen Boyd):Perfect example, diagonal plus low rank is clearly full – I mean generally it's gonna be dense, and yet you can multiply – if you know that method, you can multiply real fast. Anything else? This is classic, undergrad EE.

Student:FFG?

Instructor (Stephen Boyd):FFG. So I think – and the right way to say it – let's say it right. A would be the DFT matrix. It's N-by-N. And you would carry out the multiplication using the FFT algorithm. So, and that would be N log N again, as opposed to N squared. So that's gonna – that's an example. There's a whole bunch of others. We'll probably mention some of them. Now, we get to the bad news about CG, and this was recognized already in the '50s. It works really badly because of – with round off error. So some methods the round off error doesn't really matter, or you know, it has an effect but the errors don't build up as you go. Others, it's just a disaster. This is just, almost universally, a disaster. I mean, except for the baby examples I'll show with, like, ten variables where CG would be highly, highly, highly, highly inappropriate to use. So, otherwise, it just basically – it's almost universally, it works terribly. So that's the – out of the box, we're gonna see some tricks that will make it work reasonably. Okay, but here's a very important attribute of this method. And it's this. Depending on A, B, you know, some luck, the graduate student that put it together, put it all together – well, no, that's – then what happens is kind of cool. In a very small number of iterations, you can get a rough approximation of the solution quickly. That's a very different thing. If you go back – let's do – let's solve a circuit. I have a circuit with a million nodes. I pump in some currents. This is one step in running spice, let's say. It's a one-time step. I pump in some currents. And all I want to know is know what are the potentials there. If I use a direct – a sparse direct method, and I send an interrupt signal to it, and I just say, "Nope. Sorry, sorry, sorry. Give me the voltages." It's like, "What do you – I – it has – it's not even – have you had –." It says, "I haven't even done, like, the – I haven't finished factorizing it, so I can't give you anything."

So – but CG, if you run CG on this, it's just doing iterations, and it's just like, "Well I was scheduled – I was gonna do a million iterations. I've only done three hundred, but sure, okay. No problem. Here's V." So these are – some people call these, like, anytime algorithms. That's one name for this, which is kind of dumb. It means you can interrupt at anytime and ask – I mean well, the contract is that if you interrupt it ahead of – before it declares that it's finished, you can hardly complain about what it returns. But it returns,

like, an approximation. And, in fact, we'll see that for CG – and you'll even see why, in this case because you can analyze it. You're actually gonna get a shockingly good – and in fact, there's whole fields that are based on this where – which are actually quite funny. If you talk to people who do image restoration or they work in astronomy or something like that, they're huge and so you say, "What do you use?" "CG." And you go, "How do you actually do that?" And they go, "Yeah. We run, like, ten steps." And then you go, "Oh. Oh, that's your method?" And they go, "Yeah." And I've actually had a conversation with a person who said, "Yeah. We run ten steps of CG." You know, it's like for geophysics or something like that. So they do ten steps of CG. And I said, "Well, why not 20?" And they go, "Oh. Because each step takes like, you know, an hour." These are 3-D, big, giant, huge things, right? Okay. So, each step takes – and I say, "Well, what happens if you run it 20 steps?" And they say, "Oh, then it starts looking bad." So basically, it's already in ten steps. At ten steps – this is for a problem with, like, ten to the nine variables, or something like that, right? In ten steps, you've already gotten some vague outline of what you're looking for. But the round off is already killing you after about 20 steps, or something like that. And I'm like, "And you guys are cool with that?" And they're like, "Oh, yeah. No problem. That's fine." They were like – so that's their prescription, which is run CG until you run out of time, or the round off is built up and the results start looking worse. I mean perfectly good method. Oh, I should say, here's the joke. I'll tell you something, so you will actually, in your homework, when we finally – when we figure it out and assign it. You'll run some CG things. And, in fact, there's a matlab PCG thing, which you'll use. And you'll be very thankful for it, even though it's a five-line method. And it, in fact, when it returns, if you say run CG 50 steps, what it returns by default is actually not the iteration after 50 steps, but in fact, the iteration between 1 and 50 that had the best residual.

So – and this is not like – this is not for the reason that you see in sub-gradient method, where it's not a descent method, where ahead of time you know, this is not a descent method. This is supposed to be a descent method. It's only not a descent method because of round off things and things behaving poorly. Okay. All right. So let's look at how this works. Well, the solution, we'll call it X^* , that's $A^{-1}B$. And because A is positive definite, does it – when you – actually, there's a very useful thing. When you're solving $AX = B$, where A is positive definite, a very good way to think of it is this. You are actually minimizing a convex quadratic function. And that fits, actually, in lots of applications. For example, if you're solving $X = B$, and someone says, "Why are you solving that?" You say, "I'm going Newton's method." Someone says, "Explain to me Newton's method." Then you really realize, in fact, what you're really doing is you're minimizing this because Newton's method goes like this. You – someone says, "What's Newton's method?" And you say, "Well, I'm minimizing this weird function. I'm – I have a current estimate of X , and I go to the function and I get from a convex quadratic approximation. And I'm minimizing that." So in fact, this connection is not abstract. It comes up in the applications. Okay. Now, the gradient of this function is $X - B$, but this is also – that's a residual. So a residual in gradient of this function over here is gonna be – they're kind of the same. Okay. Now, if you take F^* is $F(X^*)$. That's this – I guess if you're – that's this thing plugged into that and – I forget what it is, but there's like an A^{-1} there, or something like that. You get an X – you get a B – it's minus B

transposing inverse – anyway, it doesn't matter. But you take F of X minus F star, here's what you get. That is – here's F of X , here, and then minus X star. And you can rewrite that this way. And so it's actually X minus X star – this is the square of – and this sub symbol here means it's the A – people call that the A norm. So that's the A norm, and it means A positive definite. But this is sort of the distance in that metric. So F minus F star is half the squared A norm of the error. Yeah.

Student:[Inaudible].

Instructor (Stephen Boyd):What's that?

Student:The B transpose X stars?

Instructor (Stephen Boyd):Yeah? You mean –

Student:It just –

Instructor (Stephen Boyd):Here?

Student:No.

Instructor (Stephen Boyd):Here?

Student:Can it be a catalytic FX minus F star?

Instructor (Stephen Boyd):Yeah.

Student:Since X and X star aren't likely to be different.

Instructor (Stephen Boyd):Yeah.

Student: B transpose X terms –

Instructor (Stephen Boyd):You mean these don't cancel away?

Student:Yeah.

Instructor (Stephen Boyd):Is that what you're saying?

Student:Yes.

Instructor (Stephen Boyd):No. I don't think – I mean unless you're really good, I don't think you can go from this line to this one by just looking at it. There's a whole bunch of lines out there commented out. No, no. I mean – yeah, look. But X star is this thing. So X star itself has a bunch of B s in it. So when I throw – this term here, when it all expands out, is gonna have a giant pile of B s as well.

Student: Can you transpose $X^* X$ transpose AX , or like –

Instructor (Stephen Boyd): Hey, that's a good point.

Student: Yeah.

Instructor (Stephen Boyd): So let me respond to your question by saying this is not – these are not meant to be self-evident. There are many commented lines out. And in fact, they may or may not be correct. They're – I'm going for like 80 – 80 – I'd go 85 percent on the reliability, maybe 90, you know? But I think they're right.

Student: Are you assuming A is positive definite then –

Instructor (Stephen Boyd): I am.

Student: [Inaudible].

Instructor (Stephen Boyd): In fact, A is positive definite until I tell you otherwise. Okay? So yeah, I mean otherwise, a lot of this doesn't make any sense at all. Okay. Okay. And now, a relative measure would be something like this, F of X minus X^* divided by F of zero minus F^* . And that would be something like that. That's τ . And it's actually the fraction of maximum possible reduction F compared to X equals zero. And actually, these things are very interesting. Here would be one. In a Newton method, what – this would have a beautiful interpretation. It makes perfect sense. So in a Newton method, it would go like this. You have a convex function. You're at a point. You form a quadratic approximation of that function, and then that goes down. And in fact, if you were to find the minimizer of that and take that step, that's the Newton step. And in fact, that height difference is λ^2 over two. That's the Newton decrement, if you remember that. So that tells you, sort of, if the quadratic approximation is right, it tells – it predicts how much you're gonna go down. If τ is a half, it says that you don't have the Newton direction, but you have a direction good enough that you will achieve one half of the reduction you would have gotten had you gotten the exact Newton step. So if you can actually calculate τ – which, sadly, you cannot – as you run these methods.

But if you could, it would be very useful because it would – because you – there's no reason you – you don't really care about the Newton direction, in fact. What you really care about is the direction that's gonna get you a good enough decrease, right? And something like if τ is a half, you could jump to the – you could quit and use that method – use that direction. Okay. So just say a little bit about the residual. By the way, I mean some of these are not too obscure. I mean so some of these multi-line things; one line should follow from the next, but not all of them. I mean without some paper and stuff like that. Okay. So this is the residual. It's the negative gradient. And you can write it this way. But in terms of this residual, you have F minus F^* . It turns out to be one-half of the residual norm, but in the A inverse norm. Okay? So in fact, in all these methods you're gonna see, a lot of things are gonna be very naturally stated in terms of the A or A inverse norm. Now, sadly, this is generally not the norm you're interested in the residual,

or something. It can be, in some applications. Generally, you're just interested in the actual residual norm, for example. And that's one of the things that's not – that you're not gonna have easy access to. But a very commonly used measure of the relative accuracy is the residual norm divided by $\|B\|$. So that's very common. By the way, $\|B\|$ is the residual norm, if you guess X equals zero. Right? Because then AX minus B is minus B , or whatever. And so basically, if this number is bigger than one, it means that's a real – that's not a really good estimate of the solution of AX equals B because in fact, you're out-performed by a function, which is six characters that returns the vector zero, which is not – that's not a good place to be. That – anyway, so that's not an impressive performance. So this is usually how this – and it scales it with the problem size and things like that. Now, you can show that τ is less than the condition number of A times ϵ squared. So – and we'll see that – well, I mean here, ϵ is easy to – you know, I mean as the algorithm runs, you have X . And you'll see part of the algorithm is you're gonna calculate the residual at each step. You're gonna calculate the X minus B . Fine. You can take the norm of that, that's easy. You know B ; you can take the norm of B . So τ , you're gonna get – sorry, ϵ you're gonna get very easily, and you'd get something like that. If A were well conditioned, this might be useful. So this is really – and this is maybe not so useful in practice, but it basically says that τ , which is arguably what you're really interested in, if ϵ is small, then τ is gonna be small too. Something like that. Of course, you don't know the condition number of A generally, so this is not that useful. Okay. Now, we get to the basic idea. So the basic idea is one of a – if has to do with Krumlauf subspace. There's – I – actually, who – anyone – who knows how to pronounce it Russian? So – because that's not how you pronounce it Russian, just to let you know. But it's been – I think it's been around for a long time. It, you know, we've been using since 1952. So now, it's Krumlauf subspace. It no longer refers to this guy's name. Okay, so – and that means that we have license to mispronounce it. So okay, so this comes up in a lot of places. A Krumlauf subspace is this.

It's the span of $B, AB, \dots, A^{K-1}B$. If you've taken a class on linear systems or something like that, it's also called the controllability subspace. And it's got other names in other fields, and I forget what they are. But it comes up in a bunch of other fields. That's called a Krumlauf subspace. And here's the cool part about it. It can be written this way. It is the set of vectors that can be represented as a polynomial of A times your vector B . Okay, where that polynomial has a degree less than K . So that – these are – it's the same thing. So you are – you're basically – you take polynomial of A , any polynomial of A up to certain degree, and multiply it by B . Okay? That's very important. It's – well, it's totally obvious because the linear combination of these is the linear combination of powers of A , then multiply it by B . Okay. So the Krumlauf sequence, X_1, X_2 , and so on, is defined this way. It's the minimum of this F of X – that's that quadratic function – over the K th Krumlauf subspace. So that's what this – that's what the – that's called the Krumlauf sequence for given A and B , right? So for example, the X_1 is something like this. It's a long – the only direction you search on is B . And so you find – you adjust the co-efficient in front of B so you get αB minimizes norm AX minus B – no, the quadratic form, okay? So that's what you – you'd minimize that along there. And you get some multiple. So now, there's many methods to compute the Krumlauf sequence. Many. But the CG algorithm is the most famous one. But there's others, so – and it's actually

worth not focusing on CG and actually focusing on the meaning of the Krumlauf subspace. So, okay. So that's the idea. All right. And now let's look at some properties. Well, your function goes down. That's – because you're minimizing a convex quadratic function over a bigger, bigger subspace. If I minimize a function over a bigger set, let alone subspace, the function value – the optimal function value can only go down. It can't go up. I mean it can stay the same, I guess. Okay. Now, if you go here all the way to N , you get X^* . Okay? So and that's actually the case, even when the Krumlauf subspace does not fill out all of our N . So it is a fact. That comes from Cayley-Hamilton theorem that the solution of $AX = B$ is in this span of KN , the Krumlauf subspace. And let's give an example. Someone give me an example of a – when would a Krumlauf subspace have shockingly – not be – when you keep running them, you get – the dimension doesn't grow? What would be a good example? Oh here's one. How about this one? How about – well, here's an extreme would be $B = 0$. So what are the Krumlauf subspaces when $B = 0$?

Student:[Inaudible].

Instructor (Stephen Boyd):They're all just a point – they're single tints. They have their only element is zero. Okay, that goes to Krumlauf's – they are subspaces, right? Every thing's cool. And if you want to minimize that you just – this sequence is just zero, zero, zero, zero. This is when $B = 0$. But here's the good news. The first one is a hit because the solution of $AX = 0$ is $X = 0$. So it's okay. A more extreme example would be something like this. What if B were an eigen vector of A ? Then what does the Krumlauf subspace look like? That's – it's just the first thing. You get B , and then you get a multiple. So the – actually, all the K 's are the same after K_1 . So $K_1 = K_2$. But again, no problem because then it says that after the first point in the Krumlauf's sequence actually solves it because you get – because it turns out, in that case, if $AX = B$, and B is an eigen vector, then in fact $X = A^{-1}B$, which is actually along the direction of B . But we'll get to that. So – but that's obvious anyway. Okay, so – okay. All right, now, this – the K Krumlauf sequence is a particular polynomial. That's a polynomial of degree less than K times B . Now, so far, every thing's actually fairly obvious. And now we get to the cool part, and it's not obvious. It's not obvious. And here it is. It turns out that to generate X_{K+1} , you can generate it from the previous two, like this. And then these are α_K . This – in fact, it's just a linear recursion. These are numbers that are gonna come up. Their particular values are gonna come up. This is not obvious at all. And this relies on A being positive definite. That's it. So in general, if I just said that's the Krumlauf – here's the Krumlauf subspace, here. And I asked you to calculate X_{K+1} , you would actually have to look back at all – I mean just in general, in fact, if A is not symmetric, you will actually have to look back over all of them. Okay? What that means is the effort to produce the next point in the Krumlauf subspace is actually not going to grow with K .

Student:So are you just taking the residual and getting the best approximation and the last?

Instructor (Stephen Boyd):No, you're not.

Student: You're not?

Instructor (Stephen Boyd): You will see what it is.

Student: Okay.

Instructor (Stephen Boyd): Yeah. It's not quite that. You'll see. So it's – these things are not totally obvious, and we'll get to what they are. So that's it. This – so if you wanna know, like, why is this interesting and all that sort of stuff, it all comes down this one statement. That's where the non-triviality is. Okay? So, okay. And just – oh, I think I mentioned this already, but the Caley-Hamilton theorem says that if you form the characteristic polynomial, like this, of the matrix, then it says you can write – if you – I mean then it says that that's zero. And therefore, will take A to B non-singular, A positive definite. So if you take all these terms – not that last one – and you pull an A out, for example, on the left. And then you put this on the other side, and you get this. You get an explicit formula. And this, I think if you took 263, we saw this. And I probably mentioned it. In fact, my forward reference was to this moment because I said there are – you know, most of the things I would say in a class like that, like we'll get to that later, we never do. I don't know if you noticed that. But in this case, we actually did, if you stuck it out this long. So, okay. So this says that the inverse of a matrix actually is a linear combination of powers of the matrix up to N . Okay? And in particular, it says that A inverse B is a polynomial of degree less than N of A times B . So it's in the N th subspace, of whatever like that. Okay? So that's the idea. So even if the Krumlauf subspace, by the way, doesn't fill out all of our N , which, of course, in general it does. If you pick a random B and a random A or whatever, it's gonna fill out everything. Okay. So that's it. Okay. So now we're gonna look at the spectral analysis of the Krumlauf sequence. So what we'll do is this. We'll take A and we will write its eigen expansion, or spectral decomposition. And λ 's gonna be this diagonal matrix with the eigenvalues. And we'll define, you know, Y is Q transposed X . That's Y expanded in the Q basis. B bar is gonna be the same thing. Y star is gonna be that.

And basically, this means that the whole problem has now reduced to – everything's diagonal. Now, you know, this is not a method. This is a method for understanding it, just so you know. So it says this. F of X is F bar of Y bar, and I just plug in this thing. That's gonna be my – this is gonna be my Y and that's Y transpose. And so you end up with this. But now this thing is completely separable, and it looks like that. Okay? So. Yeah. I mean this is a bit silly, right? It says that if you want to solve AX equals B , and someone were kind enough to provide the eigen system, you could solve AX equals B very well because it would – you'd transform it. It would convert to a diagonal system, and that – solving AX equals B , ray is diagonal, that's easy. That's no problem. And so you'd be asked to minimize this and that, I presume, you can do. And so on. You'd just get – actually, this is – that's the formula. That's basically capital λ inverse B bar. But capital λ is diagonal. So you get this. Okay? So, so far, nothing interesting here, but now let's look at the Krumlauf sequence. Well, what you're doing here is this is your – YK, is the argument of this thing over this subspace here. But this subspace here is actually really simple. It starts with a vector B bar, and you keep multiplying by these diagonal matrices.

So these are just different diagonal matrices, and what that says is that the I th component of Y at the K th Krumlauf point is a polynomial, P_K , times the eigenvalues, times $\bar{B}I$. Okay? So that's it. And therefore, we can actually say we have now a beautiful formula, not in practice but for understanding. If you understand this, then you understand – you will understand all of it, of how it works. It says this. It says that the K th polynomial – that's the polynomial that generates the K th iterate of the Krumlauf sequence, when applied to A and then multiplied by B . That's the argument over all polynomials of degree less than K of this thing. Okay? That's a positive number. It's a weight. And now you look at this and you want to know what polynomials can – what polynomials can you do here? Oh, this thing, of course, will be – you know, this will be – this thing here can be negative. That's the – that's actually the whole point, because of this thing. And so here, it's λ^I times P of λ^I , and then minus P of λ^I . So in fact, well, we'll – this is P of λ squared.

We'll get to what this means in just a second here. Okay? So you can write this another way. This – it says that if you look at how close you are to the K th iterate to the maximum reduction you're gonna get, that's $\bar{F}YK$, minus F^* . And that's the minimum of this. And now, these are just writing – reading it in different forms, so you start figuring out how you wanna – how to understand what happens. But don't worry, we're gonna go over this on next Tuesday, which, as I mentioned earlier, will probably be tomorrow. Next Tuesday. So this – if you simply work out what this is, you take out this thing here. And you end up with this. This is a generic polynomial here, whose degree is N here because – or degree is less – sorry, whose degree – this degree is less or equal to K , but it has the property that if you plug in, let's see, zero in that polynomial, you're gonna get one. That's gonna be this thing. That's the negative of this. So I can rewrite this this way. And you finally get to something like this. And it's quite beautiful. It goes like this. It says that if you wanna know how close you are to the solution after K steps, you are exactly – there's no inequalities here – you're exactly this. It says, take $\bar{B}I$ squared, that's actually how – that's the – that's basically sort of like – in fact, $\bar{B}I$ divided by λ^I is the solution, or something like that. But it says, what you want to do is you want to find – you look at all polynomials of degree K , and then this sort of a weighted positive sum of Q evaluated at those points. Okay, now, by the way, you're not supposed to be following all the details. It's just sort of the basic idea here. And then, so what it comes down to is this. It says if there is a polynomial of degree K – ammonic – I'm sorry, not ammonic, but one that adds zero as one, so constant co-efficient one, that's small on the eigenvalues of A , then it means you've got a good approximation. Okay? And this tells – now, this tells you everything. So it works like this. It says, for example, if the eigenvalues of A are sort of all over the place, but let's just see some pictures about how this works. And then we'll quit for today. Let's – here's zero. And let's draw some eigenvalues. So if the eigenvalues look like that, and you have to have a polynomial that starts at one. And if I asked you sort of what's the best degree one polynomial – it means, best, means it should be small near these points. It's gonna be something that looks like this, right? Okay? But if I ask what's the best degree two polynomial, you're gonna make something that kind of – I could – let me just try. It's gonna be a quadratic.

And it's gonna kind of look like that. Everybody got that. And look at this. On here, you're sort of – the polynomial is not too – it's pretty small. So what that says, this thing then predicts that the second – if that' were your second spectrum, the second step in the Krumlauf sequence is gonna actually have really – I mean really low error, right? Everybody see this? This is very, and by the way, if it were exactly clustered, in other words if A had exactly, like, two eigenvalues, X^2 will be the solution. Okay? That's not that interesting, but it's a fact. Everybody see why? Because then I can find a degree two polynomial that will go exactly between – that will just go right through and give you zero. And you get the answer. Does this make sense? Okay. So this is – I mean this just the first – your first exposure to it. We're gonna quit here. And in the next lecture, which like I said, will probably be taped ahead sometime tomorrow, we'll go on and look at all the implications.

[End of Audio]

Duration: 77 minutes