IntroToLinearDynamicalSystems-Lecture05

Instructor (Stephen Boyd):Oh, looks like we're on. Well, I have an announcement. Yesterday, Jacob, the TA – somewhere, lost him. Oh, well. He's around somewhere. Jacob and SEPD put together a prototype of the publicly available lectures for 263. I mean, not that you care, but just so you know, it's on the website now. You can go take a look at it, and it should be just open to the entire world, the lectures. So we haven't figured out yet how to, like, buzz your head out if the camera, you know, decides to point to you, or if we have to – I suppose we do have to; we'll figure that out later. But anyway –

We'd be interested if you find any problems with it, or if your friends somewhere else have any trouble with it, we'd be curious to – I mean, we know that it actually works from many places, but a lot of different platforms and things like that. So just to let you know, that experiment has now reached the next step. It's up and running. It's in Flash, so just like – should be not much different than YouTube. So it's up and running. We might also make available some WMV streaming, and then actually WMV download as well. We've already piddled with that internally and may also make that public soon. So – okay.

Let's continue with orthonormal sets of vectors. Does anyone have any questions about last time? If not, we'll continue. Our topic is, of course, orthonormal sets of vectors. So a set of vectors is orthonormal. So I have K vectors in RN. They're orthonormal if they're normalized. That's an attribute of the vectors separately and then mutually orthogonal, and that's an attribute of the set of vectors, and it's orthonormal if both.

Now, you can write that very compactly in matrices. It's U transpose U = I, and U transpose U = I is nothing but the matrix statement of normalized – that's the diagonal here, the ones, and the zeros on the off diagonal of I tell you that these vectors are mutually orthogonal. That's what this says. Okay. Now, last time we worked out several properties of these. We'll look at the geometric properties of a matrix whose columns are orthonormal. So let's suppose we have a set of orthonormal vectors. Oops, I said it. That was slang. An orthonormal set of vectors. After awhile, in fact, some time later today I'll just stop worrying about it.

So you U1 through UK are an orthonormal set of vectors. If W is UZ, then the norm of W is the norm of Z. In other words, if you multiply by a matrix whose columns are orthonormal, you preserve lengths; you preserve norm. Let's see how that works. It also means that people say the mapping Z to UZ is isometric. Iso means the same, and metric means that the lengths or distances are preserved because, for example, U of – if you have Z minus Z – or I should write it this way. UZ minus UZ tilde norm is equal to the norm of Z minus Z tilde.

So if two vectors in RK have a certain distance, then their images under U will have the same distance. This is very easy to show. If you look at the norm² of W, that's UZ, that's norm UZ², but the norm² of a vector is nothing but the transpose of the vector times the

vector. So I'd put UZ, and I transpose it here. I use the transpose rule to get Z transpose, U transpose U, and now I use the fact that this is the identity, and I get this, so a very quick derivation.

Now, inner products are also preserved. So if I have two vectors, Z and Z tilde, and I calculate their inner product, and then I calculate the inner product of the images under this mapping; you'll find out that they're the same, and it's the same argument. You simply say the inner product of W and W tilde – I just substitute UZ and UZ tilde, that's UZ transpose times UZ tilde. You switch it around and you get the same thing.

Now, because norms and inner products are preserved, that means angles are preserved. In other words, if two vectors have an angle of 15 degrees, then when you multiply them by this matrix U, their images will also have an angle of 15 degrees. It means, for example, orthogonality is preserved. Having a positive inner product, that means acute angles mapped to acute angles and so on. And, in fact, it really means that it's something like – well, let me careful. It's something like a rigid transformation. In R3, there are two types – I'll get to that in a minute; I'll wait on that.

It basically means that geometry, metric geometry, is preserved – distances, angles, inner products, lengths, all that. Now, a very important special case is when you actually have N of these, and these have a height of N. So I have N, a set of N – this is slang, orthonormal vectors or an orthonormal set of N vectors in RN, and they, of course, form a basis in this case because they're independent, and they span RN, and in this case U, the matrix is a square. Up till now, it's been skinny. It can't be fat; it's been skinny, a pot which includes a square as a special case.

Now that's square, and in this case it's got a name, and it's called orthogonal, and, sorry, these should have been called orthonormal, but language is weird, and orthogonal is the name given, maybe sometime in the mid-nineteenth century, and it stuck, and that's absolutely universally used, except maybe in some weird, you know, who knows. There may be people who do some weird branch of signal processing or something that people who talked about over complete bases, they might have redefined orthogonal and have some special meaning, especially they might've given it some other name.

So that's an orthogonal matrix, so it's square. It satisfies U transpose U = I. Of course, in this case, this means here that U transpose is actually the inverse of U. So it means transpose and inverse coincide. That's what it means to be orthogonal. Now, that also means that U, U transpose is I. Now, in general, it's absolutely false that if AB = I, that implies BA = I. If AB = I, it means only that B has a left inverse which is A, and A has a right inverse which is B. This absolutely does not mean, imply, that BA = I, but if they are square, it's the same. So in this case, we include U, U transpose is I.

And that's a very interesting formula; it says this: it's the sum of UI, UI transpose is I. Now, some people actually give a fancy name to a matrix of this form. This is often called an outer product, so there's lots of slang for this. I should've put this in the notes. One is it's called an outer product, and it makes, sort of, sense, right? Because the inner

product is when you reverse the two and you get a scalar. This is an end-by-end matrix, right? When you reverse them you get a scalar. It's called an outer product.

Another name for it, also very widely used is dyad. So this, in fact, when you have a sum of dyads, it's sometimes called a dyadic expansion. You will hear that very commonly, and that's the high language of mathematics. Anyone would understand what you're talk about if you say that. So this is a dyad. Oh, some people also just refer to it as a rank-1 matrix, which is actually – we'll see, in fact, soon, in fact, you already know, although we don't have the derivation, that this is the case, that it's not true that it's rank-1, just rank-1; I'll get to that later, but symmetric rank-1, something like that. These are all the synonyms for an outer product like this. By the way, if you have PQ transpose where P and Q are vectors, what is the IJ entry of this? What is it?

Student:[Off mic].

Instructor (Stephen Boyd):No, it's PI, QJ. So basically what the outer product does is it takes two vectors, actually, of different sizes, possibly different sizes, and it simply forms all possible products of one entry of one with an entry of other, and it stuffs it in the matrix. So that's what PQ transpose is, like that. Okay.

All right. Back to our main story, so our main story is that U, U transpose = I can be written this way. I mean, you could check on this. This is just one way to expand this thing is to write it out as sum UI, UI transpose. Okay. Let's see what this means. There's lots of ways to say this. If X is U, U transpose, X – well, because U, U transpose is I, we can write that this way. We can write X = sum UI, UI transpose, I = 1 to N × X, and I'm gonna reassociate things here. I'm gonna put a bracket around that. That's a scalar.

Now, a scalar can actually move to the front, and, in fact, in some cases, I guess when we recast, at the moment, that's formally a 1 by 1 matrix, but if I simply put in front of scalar, parentheses this, or parenthesis scalar to recast this as a scalar, it can move up front here, and then you get this formula here.

Now, this formula is actually quite pretty, and let's see what it says. It basically says X is a linear combination of the UI's. What are the coefficients? The coefficients actually are found in an incredibly simple way, by simply multiplying X by UI transpose. So people call UI transpose X, the component of X in the direction UI. Actually, sometimes people call this the coefficient, and you would distinguish – you would call UI transpose X the coefficient and UI transpose X × UI would sometimes be called the component of X in the direction of UI.

So here, the multiplication U transpose X, which is nothing but this, that's equal to U1 transpose X down to UN transpose X. This thing, when you make this multiplication, it's called resolving X into a vector of its UI components because that's what this says. It's basically calculating the coefficients here. When you form the operation X = UA, that's actually reconstituting X from its UI components. It's rebuilding X from the recipe; the recipe is A here.

So if you write this out, that's called the UI expansion, and you can see that it's really two operations. When you write X = U, U transpose X, if you think of it this way as U, then U transpose X, these things have real meanings. U transpose X resolves X into its U expansion coefficients. You multiply U by this vector of coefficients; this is the recipe. That reconstitutes X. So that's the meaning of this. I mean, you have to, kind of, go over this just to think about it, but there's nothing really too complicated here.

Now, this identity, which is I as U, U transpose, which is the sum. Actually, there's lot of names given to it, but in physics it's sometimes written this way, and that's mirroring this idea here. If you take X, you multiply this on the right by X like that or something like that. So I write I, and if I put this here, then you see it, sort of, completes the inner product symbol here. So I'm just curious, is anyone – you want to guess what this is called in physics? There's people here who know.

Well, that's called a bracket. So this is called – I couldn't make this up, okay? It's called a Keptbra, although I don't know how you pronounce it because this is a bracket. Well, I got news for you, that's about as fun as it gets in physics, so that's the best they can – actually, have people heard this? Cool. So they stick to it, huh? Okay. Do they think it's funny? No, they don't.

Student:[Crosstalk].

Instructor (Stephen Boyd):No, so they didn't even get it. You said when you saw it, what, did they just smile or what?

Student: Yeah, they smiled.

Instructor (Stephen Boyd):They smiled.

Student: Yeah.

Instructor (Stephen Boyd):But nothing more?

Student:[Crosstalk].

Instructor (Stephen Boyd):Okay. So let's just say they found it mildly amusing. It was, sort of, like a private joke kind of a smile or what are we – was it – what are we

Student:[Off mic].

Instructor (Stephen Boyd):Oh, other people – no, no, no. No, I'm not. I'm just asking. Okay. All right. We'll move on. Okay. By the way, you'll see this in lots of context, I presume, and already have. You may have seen the Fourier – or will see Fourier Transform described this way, and so this should look very much like many, many things you've seen, if you haven't seen exactly this. Okay.

Okay. Let's take a look at the geometric interpretation. If you have an orthogonal transformation, then the transformation, when you multiply it by an orthogonal matrix, it preserves a norm of vectors. So we already know that. It preserves angles, and examples would be something like this. If you rotate about an axis, so if you fix an axis somewhere and then rotate 22 degrees, that's a linear operation, and it's given by multiplication by an orthogonal matrix.

Another example is reflection. So in reflection you specify a hyper plane, and then you say – and then the operation is to take something on one side of the hyper plane, basically calculate the projection onto the hyper plane, and then go the exact same distance on the other side. That's a reflection. That's also linear, and it's also given by an orthogonal transformation.

Let's look at some examples in R2. So in R2, this is, kind of, pretty straightforward. If you just want to rotate by theta degrees like this, a good way to work out the matrix is quite simple. You simply figure out the first column of that matrix is the image of E1 because we know AE1 gives you the first column. So it's gonna be this thing which is cosign theta, sign theta, and, indeed, that's the first column, and the E2 rotates to the vector of -sign theta, and then cosign theta; it's the height, and that's this one, and you can check. The U theta, transpose U theta is I, and it's just a trigonometric identity. The ones on the diagonals return cosign^2 + sign^2 which will be 1, and the off diagonal ones will just cancel away.

If you reflect across this line, that's a line of theta over 2, if you reflect across it, you can work out what happens to E1 and E2, and that would give you this matrix here. It looks very similar to this one. It's also orthogonal, and this one gives a reflection, so that's the thing. Now, in fact, you can show the following: any orthogonal matrix, 2 by 2 matrix, is either a rotation or a reflection.

So what this tells you is that orthogonal matrices now, just from knowing what it means geometrically, it's gonna come up anywhere where you have isometric changes of coordinates. So it's gonna come up in areas like dynamics, aeronautics, anywhere where you're changing coordinates. GPS, it would come up all the time. It's gonna come up in navigation, and it will come up in robotics all the time.

So it'll just be matrices filled with cosigns and signs, and that'll be the orthogonal transformations that map, you know, the coordinate system of the third link in the robot arm back to the base coordinate system, or in GPS, you'll have the world fixed – whatever the earth-centered, earth-fixed coordinate system and some other coordinate system, and there'll be rotation matrices, orthogonal matrices that affect the transformation. Okay? So you'll see them in lots and lots of fields like that. Okay.

So now we're gonna look at something called the Graham-Schmidt Procedure, and before we start, let me point out where we are in the class and where we're going. So this may be a time to have a little road sign to say where we are and where we're going. So far, we've rapidly reviewed these ideas from linear algebra that you may or may not have

seen before, but, at the moment, you actually have, so far, we have introduced no computation teeth into what we're talking about here, right? So, so far, you have no idea how one would actually, sort of, compute a subspace, how would you calculate the range? So, in fact, so far it's all been talk.

That's gonna change in the next two lectures, and it's not a big deal, but it's gonna change in the next two lectures, and all of the stuff we've been talking about will be given algorithmic and computational teeth. So you will actually be able to do things with all the things we've been talking about, which actually will make the course a lot more interesting, I assure you, for me, anyway. It's gonna be a lot more interesting and a lot more fun. So that starts now. That's where we're going.

Okay. Graham-Schmidt Procedure, this you probably have seen somewhere. It's not exactly a very complicated thing, so that's okay. Graham-Schmidt Procedure goes like this. It's an algorithm, and it accepts a set of independent vectors. Ooh, that was slang. I really have to work on this, right? Because independence is not a property of vectors individually; it's a property of sets of vectors. Okay. I'm gonna try it again, and then I'm just gonna say forget it. I gotta collect my thought.

It's given an independent set of vectors. Yes, it came out. So, actually, you could help me, and whenever I slip into slang you could hiss just a little bit or something like that just to help me here, but after awhile you just get used to it. All right. So given an independent set of vectors – yeah, that's good, okay.

A1 through AK and RN, Graham-Schmidt procedure actually finds a bunch of orthonormal vectors with the following property. They have the same span as the vectors you started with, and, in fact, that's true no matter where you stop the algorithm. So if you look at the Q1 up to QR, these are orthonormal, and they all have the same span as A1 through AR. Okay.

So what this says is that Q1 to Q4 is an orthonormal basis for the span of A1 through AR, and the idea of the method is, first, what you do is you orthogonalize each vector with respect to the previous ones, and then you normalize it to have norm 1. I mean, it's not particularly complicated. So let's see how it works. So it's actually quite straightforward. The first thing you do is you're given A1, and your task, or at least the specifications, are to produce – well, a Q1, which has the same span as A1 and should be orthogonal. Well, it should be a singleton which is orthonormal which is the same as saying it should have length one, okay?

That's easy to do. You simply take A1 and you divide it by its norm, and that gives you a unit vector that points in the same direction as A1, and we'll do that in two steps. We'll say Q tilde 1 is A1, and then Q1 is Q1 tilde divided by Q1 tilde, norm. This is the normalization step. This is the orthogonalization step except there's nothing to orthogonalize against.

Now it gets interesting in Step 2. You take Q2 tilde is A2 minus Q1 transpose A2 × Q1. And this is what you do, now when you subtract this, you are actually – the name of this step, if you asked me to describe it, is this: You are orthogonalizing with respect to Q1. That's what you're doing in this step. You're orthogonalizing with this – you can check that Q1, Q2 transpose, Q2 tilde is zero, and you do it by subtracting the appropriate component of Q1 – the appropriate multiple of Q1, okay? So that's what this does.

This ensures that Q1 transpose, Q2 tilde is zero. That's what it does, and you can check because Q1 transpose, A2 is the same as this. You could work out what this is times – and then you have Q1 transpose. Q1 transpose, Q1 is one; it's normalized and so is zero, okay?

So this is really the orthogonal, in fact, this is called orthogonalization, okay?

Then when you orthogonalize, there's no reason to believe that what you are left with has unit norm, so you normalize. So basically it alternates between orthogonalize, normalize, orthogonalize, normalize. Now, you always orthogonalize with respect to the stuff you've already done, and that involves this, okay? So this is the next step is you simply orthogonalize and so on, and you keep going.

And there's lots of questions you have to ask here. In fact, what you have to ask in an algorithm like this is if you fail, why? And you'd have to argue why you don't fail. If I failed here, if I had a divide by zero exception here, what would it mean? Suppose here we fail, as we really can't fail here. Here you can fail by a divide by zero exception. What would it mean?

Student:[Off mic].

Instructor (Stephen Boyd):A1 is 1; do you have any problem with that? It violates the contract here because we were allegedly given or passed a set – ooh, an independent set of vectors. Caught myself that time, okay? If A1 were zero, that contract was violated that we were not sent an independent set of vectors, okay?

Now, the second one is actually really interesting. What would happen if we – oh, not here, here you go. What if there is a divide by zero exception at 2B; what would it mean? It would mean that Q2 – well, it'd mean Q2 tilde is zero. That means A2 is equal to this thing times Q1. Q1 is also a multiple of A1. That says A2 is a multiple of A1. Once again, it means this – you get a divide by zero exception only if A2 is a multiple of A1. That means the pair, A2 A1, A1 A2, is not independent. It means that the preconditions didn't hold. Okay.

By the way, some people use – you can use a Graham-Schmidt-like procedure to check, in fact, if a set of vectors is independent. That came out right, by the way. It checks because if it has a divide by zero exception somewhere, instead of, sort of, jumping core, it can throw an exception saying – in fact, it can be quite explicit. It can actually say these are not independent. In fact, it can say the following: The last vector that I processed is

actually the following specific linear combination of the previous ones. So it actually returns a certificate proving that the set of vectors is not independent. Everybody see what I'm saying here? So these are the methods, and we'll see how that works.

All right. So here's the geometry of it. You start with A1, that's over here, and that's also Q1 tilde. The only thing here to do is to divide by the norm, so apparently that's our unit length. The next step is you take A2, which is like this. You subtract off the Q1 component, that's you subtract off this. This vector is this thing here, right here. I subtract that off, and what I'm left with is this. Oh, by the way, there's a name for this. Some people call that The Innovations or something. In a statistical context, in signal processing context, it's called The Innovations, this thing.

I mean, it makes sense. This is, sort of, what's new in A2 that you haven't seen already in A1; it's what's new. There's other reasons why it's called The Innovations, but, I mean, that, sort of, makes perfect sense even in this geometric context. So that's the innovations, and you simply normalize the innovations. So this is the picture.

Now, if you do this for awhile – if you run up our steps, you have the following: At each step here, I take these terms here, and I put them on the other side, and you see something very nice which is that A3 is equal to Q3, which is, in fact, nothing but – Q3 tilde is going to be Q3 × norm Q3 tilde or something, and times a scalar, and you see that this expression here gives A3 as a linear combination of Q1, Q2, Q3.

Oh, by the way, that's, kind of, what we're required to do. This says that at each step, you express AI effectively as a linear combination of Q1 up to QI, like that. And we're gonna give these names. We're gonna call these R1I, R2I, and RII, okay? And they just come right out of the Graham-Schmidt Procedure. And RII is not zero because RII – in fact, not only that, you can even say this. RII is positive because it's actually this thing, okay?

Well, if you write this set of equations out in matrix form, let's see what it says. It says that each AI is a linear combination of Q1 up to QI, and if you write that on the matrix form, it's an embarrassingly simple formula. It's four ASCII characters; it's A = QR. So A = QR. Here A is RN by K. That's a matrix; you've concatenated the vectors into a matrix. Q is of the same size; it's N by K, and R is K by K, but what's interesting – oh, let's see how this works out.

This says A = Q × R, and you can check that by the rules of batch, batch matrix multiplying, and if you interpret column by column, it says basically that A1 is this thing times that, and that's Q1 × R11, exactly like we said. A2 is this thing times the second row, and that's gonna be Q2 × R12 + – Q1 × R12 or R12 + Q2 × RQ2. These are scalars, so they, kind of, they rotate forward. That's the picture.

The lower triangle of – the fact that this is upper triangular here comes from the fact that when you apply this procedure, you express AI only as a linear combination of Q1 up to QI. It doesn't involve QI + 1, QI +2, and so on. There's, sort of, a causality here, which is, sort of, what you should think of automatically when you see an upper triangular

matrix or a lower triangular. Suitably interpreted in the meaning is something like causality here.

And, in fact, I guess I can say what the causality is here. This algorithm, the QR algorithm, spits out Q7 before it has even seen A8, A9, A10. Those may not even exist. That's the causality, that when you calculate Q7, you only need A1 through A7 to calculate Q7, not A8, not A9, which may not even exist. So that's the causality that you have here. Okay.

Now, it's a beautiful decomposite that says you can write a matrix as a product of a matrix whose columns are orthonormal – that was slang – times a lower triangular matrix which is actually invertible, upper triangular matrix is invertible. And this is called the QRD composition, and it's also, maybe, sometimes called the Graham-Schmidt Decomposition or something like that, but this is it. Okay.

And that's called a decomposition or a factorization of A, and then you'll find, actually, in terms of English, there are two verbs. I don't know which is right, but you will say that the verb to do this is either to factor or to factorize, and I think it depends on whether you, your advisor, and your advisor's advisor were, maybe, educated, like, at Cambridge, or Oxford, or something like that. I'm not exactly sure, but you see it about half and half. People will say, "You should factorize QA to get QR or factor." I think, actually, factorize is winning out now, so – all right. So that's called a factorization. We'll see several factorizations before the class is over. Oh, by the way, I've now shown – by the way, we have a big stack of unproved things, and we just popped one, for the record.

Earlier I said that if a matrix – well, not quite; we didn't get that. This is one of – this is – well, okay. No, we didn't. Sorry. Scratch all that. So it'll be fun. When we can actually edit the videos it's gonna be fun because I'll start and then my head will go like this, and it'll be a little bit shorter. It'll be good. I can't wait to do that. All right. Forget all that. Let's move on.

By the way, the method I showed here is – the Graham-Schmidt Procedure the way I showed it is actually – and obviously it's completely correct. It is not the way it's actually done in practice. In fact, people use a variation on it called Modified Graham-Schmidt Procedure, which is less sensitive to numerical rounding errors, and, in fact, they don't use the Modified Graham-Schmidt Procedure either. I mean, if you really want to know how it's done, it's done by blocks and things like that.

So it's fairly complicated, the same way, I think, at one point, when I was ranting last week, maybe, about how do you multiply two matrices. Anyway, it's not the four line C program. It's actually done by blocks and things like that, same for QR factorizations. If you were to look at the actual code that runs, for example, in Matlab, if you do a QRD composition – and let me point something out.

Math Works had nothing whatsoever to do with any of the numerics. All of the numerics in Matlab – just to make sure this is absolutely clear because it irritates me beyond belief

when people think otherwise. Every numerical method in Matlab relies on open source public domain software written by professors and graduate students, just so you know. So please never say, “Oh, the numerics in Matlab, beautiful.” Because you, and your friends, and your professors, and colleagues, and peers wrote it, and it’s free for everybody. Sorry, just had to have that slip out. Okay. All right.

Okay. So what happens is this is actually an algorithm for computing an orthonormal basis for the range of a matrix. I mean, that’s what it does; it creates an orthonormal range for the basis. By the way, as a method right now, it doesn’t really – you’ve got a little bit of algorithmic teeth. A modified version of this would actually allow you to check if a skinny matrix is full rank. It would allow you to check if a set of vectors is independent. How would you do it?

Well, using this method, as I said, you would modify the algorithm to not dump core on a divide by zero exception, but instead, to return something saying they’re not independent. That’s what you would do, okay? So at least you have a little bit. You have an algorithmic method to check the rank – I’m sorry, to check if a matrix, so far, is full rank. We’re gonna get to rank in a minute.

Now, the General Graham-Schmidt Procedure – General Graham-Schmidt procedure works like this. In the one I just described, we assumed the input set of vectors is independent. Notice that’s slang in writing, but you’re allowed, by the way, to write slang on lecture notes, not in papers, by the way – oh, unless it’s obvious. Unless it’s in a paragraph in which it’s clearly, kind of, fuzzy and giving the idea, then you’re allowed to use slang, but in lecture notes I’m allowed to use slang.

Now, if they’re dependent, what’ll happen is you’ll get a divide by zero exception in the algorithm we just looked at. Now, here’s the modified algorithm. It’s really simple. It does this – let’s go back to the algorithm and see what we do. Here’s what happens. What happens is this. You do the same thing as before. So, for example, you remove the previous Q components from the current vector, all right? Now you attempt to normalize. The only thing that can fail here is that Q3 tilde can be zero.

So you simply put a test in front that says if Q3 tilde = 0, by the way, that has a very specific meaning. That will only happen – it means, specifically, that A3 is a linear combination of Q1 and Q2, which is the same as saying it’s a linear combination of A1 and A2. That’s the only way this will fail. If it fails, you simply fail to do this and you actually just get – you say get me another column. You just say, instead of dividing, you call get call or get next vector, and you pull another vector in, and you do the same thing; you normalize. You didn’t spit out a Q.

So this is the algorithm. It runs like this: U form AI tilde is this thing, so you actually orthogonalize with respect to the previous things here. If there is any innovation because A tilde you can interpret as the innovation. It’s basically, sort of, what’s new that I haven’t seen already? That’s A tilde. If this is zero, you just do nothing. You just go right

back, and you get another column, and you keep going. You don't generate a QI. Otherwise, you generate a Q.

So the number of Q's you generate actually is gonna tell you something about the rank. Okay.

Now, by the way, this works perfectly nice. Let's see how this works just for fun. All right, let's do it this way. Suppose I put in vectors that look like this: 0, E1, 2E1, and E2. Actually, you know what? It doesn't matter. I'm gonna call that A1, A1, and A2, and A1 and A2 are independent. What does a Generalized Graham-Schmidt do here? Well, it checks zero. It orthogonalizes it with respect to the previous Q's of which there are none, and it attempts to normalize.

You're gonna have some serious trouble normalizing zero, so this doesn't happen; this fails. So you simply skip over it, and you get a new vector which is A1. You orthogonalize with respect to previous Q's; there are no previous Q's, so you are already orthogonalized, and now you normalize with respect to A1. So we divide A1 by its norm, and that's gonna be Q1. And so notice that the span of the first two vectors here, that's zero and A1, is now, in fact, Q1, which has dimension one, okay?

At the next step you get 2A1. You orthogonalize, and you'll get zero because 2A1 is in a linear combination of A1 – well, of A1. So here, once again, this fails again, and you don't spit out a Q. You pull an A2, and then you get a Q. So as you process these four vectors, you will, in fact, generate only two Q's. Reason, the rank of this is exactly equal to two, okay?

So you can actually say much more. You know, it's not here. It's, kind of, silly, isn't it? Oh, it's not Q1 through QI; they're indexed by a different way. Q1 through QR is an orthonormal basis for the range of A, but notice what happened here. R can be less than K, okay? So that's what happens. R can be less than K. Oh, there's another name for this. This thing is sometimes called the Rank Revealing QR Factorization.

So that's a great name for it. Rank Revealing QR, and you will hear this. People will talk about a Rank Revealing QR Algorithm. That's basically a Q algorithm that's modified like this, doesn't dump core when what you pass it is not independent, but an interesting thing is when it terminates, it has actually calculated the rank; it's got the rank.

But there is one thing, by the way, that we're not looking at and, in general, don't care about in this course. They're important issues. It turns out that you have to be a little bit, you know, when you have numerical codes, this is not A tilde = = 0. In other words, it's actually exactly zero as a double vector. Now, more likely than anything, this is something like the norm is less than some tolerance which is really small, okay?

So whenever you see something like a rank revealing QR, be forewarned that there is a tolerance in there, and that you may have access to it. You may want to change it. You may want to know what it is. It is probably set very low, so low that, for engineering

purposes – it may be inappropriately low for engineering purposes, but I mention this just because it's an issue. You really don't have to worry about it, but you should be aware that there is a tolerance in a real rank revealing QR algorithm. Okay.

Now, in matrix form, you can write it this way. You can write A = QR where Q transpose Q is I. Now, you should simply process these five ASCII characters. You don't need the R because you're sophisticated now. You know that's just simply the width of Q. When you see Q transpose Q = I, that should simply be an alias or a macro for the columns of QR orthonormal. So that's what this is.

And R is an R by K, and it looks like this. The R's get spit out this way; it's quite interesting. I think it's called Upper Staircase or there's another one you'll hear like Upper Echelon or some – I don't know. You'll hear various names for this type of factorization – upper staircase, upper – I don't know. Who knows? Reduced Row Echelon – I don't know what it is. Anyway, but it looks like this.

Now, the corners on the steps occurred exactly when you spit out a new Q, which occurred when the orthogonalization, the innovation, turned out to be non-zero. So for this picture, we can say a lot about the rank of various bits and pieces. So I'm gonna ask you some questions. The fact that a Q1 was generated on the first step is equivalent to what statement about the columns A1 up to AK? We generated a Q1 on the first step, why?

Student:A1's not zero.

Instructor (Stephen Boyd):A1's not zero. We generated a Q2 on the second step; what does it mean?

Student:[Crosstalk].

Instructor (Stephen Boyd):A1, N2, H over independent. Okay. On the third step, nothing was generated. It has a meaning; what's it mean?

Student:[Crosstalk].

Instructor (Stephen Boyd):Okay. A3 is the linear combination of A1 and A2, exactly. Okay. So you get the idea. Okay. So actually the structure of this is quite revealing. It, kind of, tells you – well, it certainly tells you in ascending order, it, sort of, tells you which vectors are dependent on previous ones, and so you get a lot of information and structure here. Okay. Oh, I don't know. Let's see. It's only count out a couple. What is the rank of A1, A2, A3, A4?

Student:[Crosstalk].

Instructor (Stephen Boyd):It's three, and, in fact, we have an orthonormal basis for it; what is it?

Student:[Crosstalk].

Instructor (Stephen Boyd):Q1, Q2, Q3 – Q3, I heard Q4, but I'm just ignoring it, okay. That's it. It's Q1, Q2, Q3, right. Did I do that right? Yes, I did. All right. It's okay. Okay. Now, there's another way to write this and another way you'll actually find this – this is one method. What you can do is actually you can permute the columns. Now, permuting the columns is the same as multiplying by a permutation matrix on the right.

I used to get all confused about this, and I still do, sometimes, but I just go – actually, the one thing you do actually if no one's looking, you can write down a 2 by 2 matrix and do this secretly, but not if anyone is looking. Don't ever do that because it's – well, it's humiliating, really, if someone catches you. I'm talking about whether permuting columns or rows corresponds to permutation multiplication on the left or right.

The other way to do it is say well, look, columns are associated with inputs; you know that. So, basically, if you mess with the inputs, that's something that should happen first before you apply the matrix. If it applies first, it's on the right. I can see this was extremely unhelpful. Well, okay, fine. Make up your own mnemonic. I don't care. Okay.

So I take this matrix here, and I apply a permutation on the right that rotates these columns forward, right? And takes all the columns when there were no innovations and rotates them to the back. If I do that, it's gonna look like this: A is Q, R tilde, S – that's, kind of, whatever's left over – × P. Now here, Q is, again, a matrix with columns that are orthonormal. R tilde is upper triangular and invertible, so now it looks like this. It's just like the old QR factorization; that's that. S is – well, who cares what S is, and it looks like that. So you have this picture. Okay? And then Q goes here, and A is the same size as Q, okay? So that's the picture. Yeah, you've rotated everything forward – or rotated the ones where you had something. Okay. So this is another form you'll see this in. Okay.

All right. Let's talk about some applications of this. Well, it directly yields a orthonormal basis for the range of A. Now, I'm gonna go back. Remember, I went on a little detour, a little ahead of time. Now it's actually time. So, now, we're gonna actually pop something off our stack of unanswered questions, or something like that, or unproved assertions.

Earlier in the class, there was a factorization interpretation of rank, and the factorization went this. It says that if a matrix A has rank R, then you can factor it as a matrix, you know, you can make a tall, skinny factorization, okay? So you can get a tall, skinny factorization where the intermediate dimension here is the rank. I asserted that without proof, and we also had interpretations of this right then. Okay.

Now you have one because here is one, right there. So that gives you one right there. That's exactly the size. Here's A is Q, that's the first part of the factorization, that's what I called B, and you call this thing C, and it's got exactly the right sizes, so it gives you that factorization.

By the way, this means – and you can do, sort of, a pseudo application. Earlier we found out that a factorization of a matrix, especially if it's a stunning factorization, like if it's 1,000 by 1,000 matrix, that, for example, has rank 15, and it factors as 1,000 by 15, and 15 by 1,000 matrix.

We found out many things you can do now. One is you can multiply by that matrix super, way faster than you could before, okay? And let's not make it 1,000. Let's make it a million, million by million matrix, okay? So if A is a million by a million, it says that you can do this – it says you can factor it this way. You couldn't even store it, a million by million matrix, let alone multiply by it, okay?

Factored as a million by 15, 15 by a million, it's no problem and would be shockingly fast. I can do that on my laptop. So, now, the question is you're given such a matrix. Of course, you can't store it, so the story gets a little bit fuzzy here, and you're asked to see if you can do such a factorization. You can apply something like this Modified Graham-Schmidt, and you can keep getting columns and stuff like that.

By the way, you don't actually have to ever store a whole lot when you do this algorithm, right? All you need is a method to get – you just request columns, and you check against the other one. You could basically say you know what? If the ranks more than 20, forget it; this idea is not gonna fly. So you keep processing the columns, and you see how many Q's you pick up. If, in fact, you process all the columns, and Q only got to 15 or 16, you win. You have a factorization, low rank.

That work was an investment. You can now multiply that matrix by that matrix some absurd about of time faster. If that's a simulation, or some signal processing calculation, or something like that that's gonna happen a lot of times, you just won big. By the way, this is not so useful because these are exact factorizations. Later in the class, we're gonna see approximate factorizations. That's much more powerful, is an approximate factorization, but for now, we'll leave it this way.

Student:[Off mic].

Instructor (Stephen Boyd):No, it doesn't. No. I'll repeat the question. In Matlab, when you multiply big matrices, does it attempt to do anything like this? And the answer is no, it doesn't. In fact, I don't really know any linear algebra system, although one could be developed, that would do cool stuff like that. That would basically flag a matrix as low rank and – now, ideas involving rank are widely used in numerical linear algebra, but, as far I know, there's no general system. It'd be quite cool to build one, actually. Okay.

So let's see. We've already checked a couple of things. Oh, here's one. If you want to check if a vector is in the span of a bunch of matrices, which is basically saying, for example, suppose you want to check if B is in the range of A, no problem. All you do is you could run the Modified Graham-Schmidt on A1 up to AK and B, and what you check on is this. You only check that when B was pulled in, you generated a new vector. If you did generate a new vector, a new Q, that means B is not in the range. If you didn't – if on

the last step, you didn't generate a new Q, then B is in the range of the previous ones. Actually, the algorithm automatically solves, by the way, AX = B in that case. It finds an X that gives AX = B.

Now, one interesting fact is that because of this causality, the Graham-Schmidt Procedure yields a QR factor is A and yields QR factors that if you stop it early, and if you send a terminate signal to the QR process, and it stops, it actually has calculated a QR factorization of a leading – if it's processed P columns, it's the leading sub-matrix of A, has already been processed.

So what's, kind of, cool about that is this. It says that when you actually calculate, for example, the QR factorization of A, you've actually simultaneously calculated the QR factorization of every sub-matrix of A. I have to say, it's not every, but it's every leading sub-matrix. So here's A, and I'm gonna call a leading sub-matrix, what happens if I truncate this way, okay? And you'll actually have – if you truncate Q and R appropriately, in the same way, you get the QR factorization of a – oh, I don't know. What you call that; what would you call the first chunk of a matrix? What would you call it?

Student: The head.

Instructor (Stephen Boyd):The head of the matrix? Sure, yeah. Yeah – no, no, no, that's fine. Okay. What do you call it when you have a string at the beginning of a string?

Student: Prefix.

Instructor (Stephen Boyd):Pre – what, prefix? Prefix of a matrix; what do you think?

Student:[Crosstalk].

Instructor (Stephen Boyd):Head, prefix? Listen, we gotta step in. If we don't, the physicists are gonna step in, and it's gonna have some totally idiotic name, so – sorry. That was just for you. Yeah, I presume you're in physics?

Student: No, I was.

Instructor (Stephen Boyd):You were.

Student: Yeah.

Instructor (Stephen Boyd):Okay. What are you in now?

Student: Aero.

Instructor (Stephen Boyd):Aero, all right. I have plenty to say about those people too, sorry. You had a question? No, okay. All right. So prefix or head at the moment. I'm just

saying we have to be involved because, otherwise, you won't believe the names other people will assign to them. So all right.

I'll mention one more thing which is the full QR factorization. Full QR factorization is this. You write A = Q and R1. Let's call that the QR factorization we just worked out, without the permutation, just assume it's full rank or whatever. It doesn't matter. Actually either way it doesn't matter. You write Q – here you put Q1, and what you do is you fill it out to make it square, okay? And you put a zero here, so it really, I mean, it doesn't matter, and let me explain a little bit about how you do this.

This is a very common procedure. It basically says given a, like, let's say K orthonormal columns, that was slang. I'm gonna stop saying that, but anyway. Given an orthonormal set of K columns, the question is – and these are in R10. It's very standard to fill it out, which is to generate something like a Q8, Q9, and Q10 that makes the whole thing an orthogonal matrix, okay?

Now, it sounds weird. The question is how do you do it, and the way is actually quite straightforward. What you do – let me just show you how you would get this Q2. It's really, kind of, dumb. It works like this. Let's feed in to our rank revealing Q4 the following: A and then the identity, okay? So that's what we're gonna feed in. Now, that matrix, sure, it's definitely full rank because of this I here, okay? No doubt about it, that's full rank.

But what we're gonna do is I just want to set a flag that the first time our QR algorithm gets a column, has to pull into the I matrix to get a new column, I want to set a flag. So here's the way it happens. Our modified QR starts working on A, pulls in column one, processes it, two, three, four, five. Let's say it finishes up A, and it doesn't matter what the width of A is. Well, let's just put some numbers on here for fun.

That's ten. Let's say that A is, you know, seven vectors, and let's say I've run this Graham-Schmidt QR Factorization, or Graham-Schmidt, or algorithm, or whatever on this, and let's suppose the rank of A is six. So what will happen is I will have pulled in A7 and processed it, okay? And how many Q's will I have at that point? Six, I'll have an orthonormal basis for the range of A. Okay.

Now, normally that's where you stop. That's where our old factorization stops. Now we're gonna do the following though. I'm just gonna go get E1 because that's the next thing here, but I'm gonna set a flag, which is by I've overrun – I'm indexing out of my matrix, okay? I'm no longer processing A. I'll take E1. Now, E1 might be in the range of A, in which case, I won't generate a Q7, okay?

No problem if it isn't there. I pull E2 in, and I process that, okay?

Now, when I finish here, I must have Q1 through Q10. I absolutely must, why? Because the rank of this matrix is 10. So, however, the last three, Q8, Q9, Q10 were not generated from A; they were generated from this I or whatever other matrix you put here, okay?

However, the Q8, Q9, Q10, they're all normalized. They're mutually orthogonal, and they're orthogonal to Q1 through Q7, and I'm gonna write it this way.

So this would be that first – this is the chunk of Q's that I got out of A, and this is what happened when I actually – I called get call on A, and it returned an EOM. I guess that's end of matrix flag, okay? So it returned an EOM, and I said no problem, and I got a column from – I just got E1, let's say, and kept going. So that's what this is, okay? So this is how you do it, and this is very cool because this is finishing off, extending an orthonormal basis from a given basis out to a basis for the – a basis for a subspace to a basis for the whole thing.

I should add, all of this, at the moment, since we're not looking at applications, it probably doesn't make a whole lot of sense, or, even if it made sense, is not that interesting. It will be very interesting when we actually start doing stuff. There was a question?

Student:[Off mic].

Instructor (Stephen Boyd):E2, you go to E2. Then you go to E3, but the point is, by the time you finish processing this, you have to have ten because the rank of this matrix, I guarantee you, is ten no matter what is, including A = 0. That's about a low as ranks go, okay?

So in A = 0, let's find out what would happen here is A would generate no Q's anywhere. So you'd stop. Then it would be correct. It'd be, like, an empty matrix or something like that, no Q's, okay? Then the whole thing – then I would actually, in fact, be Q2 because they're already orthonormal, and in that case, this would be zero – not zero, sorry. No, and Q2 would be I. Does that answer your question? Okay.

So this is called extending an orthonormal basis. Now, the two subspaces, the range of Q1 and range of Q2, they're actually very interesting. They're called Complimentary Subspaces. They do two things. First of all, they're orthogonal. We overload – right now, you know what it means to say X perp Y. It means X transpose Y is zero. They have zero inner product. That's what that means. We overload the perp to sets of vectors.

So you can say a set A is perpendicular to a set B, and the meaning is this. Any vector in A is orthogonal to any vector in B, okay? So here, you would say that range of Q1 and the range of Q2 like this are orthogonal like that. It means that anything in here and anything in here, and, by the way, you need to check that. You really need to check that; you can check it.

It means that anything in here is a linear combination of, let's say, Q1 has got columns Q1 through QR, and this has columns qR + 1, that's little q, you can tell by the way I say it; it's a little bit softer, the lower case q. It's, like, qR + 1 up to qM. This is a linear combination of the first R1's, and it's a linear combination of the others, but they're all mutually orthogonal, so the inner product is zero. Okay.

Now, the other thing is we overload sum for sets. When you overload the sum for a set of vectors, it means the following. It's a new set, and it consists of all sums, all possible sums, of a vector from this set plus a vector from that set, okay? So we're gonna overload plus as well to sets of vectors, okay? Now, so you write this this way. If you do that, you'd say that RFQ1 + R of Q2 is equal to RN, like that, and what that says is the following. It says if you take all possible sums of a vector from range of Q1 plus all possible – well, if you pick a vector from here and here, add them, and then do that over all possible choices, you get RN. Yes, a question?

Student:[Off mic].

Instructor (Stephen Boyd):What's what?

Student:[Off mic].

Instructor (Stephen Boyd):The bar, you mean this thing?

Student: Yeah.

Instructor (Stephen Boyd):I'll tell you what that means in a minute. Okay. So this is what people would write. And, by the way, the way you'd say this in English would be something like this: oh, range of Q1 + range of Q2 is all N, or you'd say something weird like together they span RN. That would be the English phrase. That's how you'd say that informally.

Now, this bar, this is a symbol that says R1 – this terminology means R1, the range of Q1 + the range of Q2 is RN. That's what the plus means, and that little perp, it saves you writing this: And range of Q1 is perpendicular to range of Q2, so all the perp means is that. You will see it. There's no particular – it doesn't save you a whole lot of trouble to write out both, but you have to admit, it looks cool, and it's a very advanced notation, and if you do that, people won't know what you're doing, and you can say – well, you'll see I'll give you some words you can say, and your roommates or whatever will be very impressed. So you asked what it's for; that's basically what it's for. Okay. All right.

So another way to say that is this. In this case, you say that they're orthogonal compliments, the two subspaces, and another way to do it is the perp symbol actually acts as a post-fix superscript on a subspace, and it is, in fact, the orthogonal compliment. It's the set of all vectors orthogonal to every vector in this set here, okay? We'll have a homework problem or something on this soon to, kind of, help you get straight about this, and we'll look at some more of this in another context.

So now, I want to say one more thing. This is, at the moment, gonna look a bit bizarre, but later it'll make sense, I promise. If you do a QR factorization of A transpose, now A and A transpose have exactly the same rank. So I do a QR factorization of A transpose, I get something – sorry. If I do a QR factorization of A, which is a full one, and I transpose it, I get this. Sorry, I should've said A is Q1, Q2 × R1 N0, like that, okay? That's the full

QR, and I transposed this QR factorization, and you find the following. You can now imagine what A transpose does to a vector by first operating here and then operating here.

Now, this is interesting because basically it says that when you form A transpose Z, you first for Q1 transpose Z and Q2 transpose Z. So you produce two sub-vectors. The first one – let's see if I've got this right. No, I've got it totally wrong, completely and totally – there we go. All right. So it's really this. Q1, Q2 – let's try that again – is U form R1Z and zero here. When I multiply – no, no. I had it completely right.

I'm wondering if this posting the lectures on the web is such a great idea. We better look into editing them pretty fast, I think. That's my conclusion, okay? This happened just because they've been posted. All my friends at MIT will be watching this, chuckling. I'll get you, don't worry. Okay. Let's try it again.

Okay. We're just gonna go slow here. R1 transpose zero, okay, there we go. A transpose Z is this. There we go. So the first thing we do is we calculate Q1 transpose Z and Q2 transpose Z. Q1 transpose Z gets then multiplied by R1 transpose and actually causes something to come out. Q2 transpose Z then gets zeroed out. So you calculate Q2 transpose Z, but then nothing happens, and now you see an amazing thing. You see that basically A transpose Z = 0, actually if, and only if, Q2 transpose Z = 0, which is the same as saying that Z is in the range of Q1.

Student:[Off mic].

Instructor (Stephen Boyd):Where? Thank you. It's just not – this is what happens. I knew we shouldn't have done it, see? Okay. There we go. Thank you. There we go. All right. Okay. Now, what this says is interesting. It says that the range of Q2 is the null space of A transpose. It's the null space of Q1 transpose, which is the same as the null space of A transpose, okay?

And what you find out is that this second – when you fill out this second part of the QR factorization here, this Q2, what it really is is the columns are an orthonomal basis for the null space of A transpose, okay? And what you include is this, the range of A and the null space of A transpose are complimentary subspaces, okay? Indeed, one of these is the span of Q1; it's the range of Q1, and the other is the range of Q2 obtained in this full QR factorization. So they are complimentary subspaces.

That means, again, this is the method used to impress your roommates and so on. It means that they add up to all of RN, and it's says that they're also orthogonal subspaces. By the way, one consequence of this is the following, is that every vector in RN can be written uniquely as a sum of a vector which is in the range of A and a vector which is in the range of A transpose, and more over, those two vectors are orthogonal, okay? That's what it means.

And so people call this the Orthogonal Decomposition of RN induced by the matrix A. That's the name for this, and once you know this, and we have, now, a constructive

method for doing this. In fact, we simply apply the Modified Graham-Schmidt to AI, that's the algorithm. If you do that, you can generate this Q1 and Q2, and all of this will actually – and now it's constructive.

Now you can actually prove most – I think maybe actually all of the assertions from the linear algebra review once you know this. It'll actually be a little bit easier in a couple of weeks when we do more material, but this is it. But if you switch A to A transpose, you get a decomposition of RK, which is the other side, and you get this, the null space of A is an orthogonal compliment of the range of A transpose. And so these are the various – these are, by the way, the four fundamental subspaces. There's whatever people glorify them as, these four.

Actually you don't have to worry about them too much. All that really matters is what they mean in the context of applications, I think. So that's the importance, so you don't have to – these are just silly symbols that are interesting. It's much more interesting when we start doing estimation, or channel decoding, or input design. Then these things are very interesting, and they have very real consequences, and that's what we'll be doing in the next bit.

Let me show you a couple of things here. When you show your friends – when you assert this they say, "Wow, that's a – " oh, by the way, some people consider – I think this is sometimes called – this is also sometimes called one of the – this is also one of the major "results" in linear algebra. So it's a big deal. It's treated with a lot of reverence. You're never supposed to joke about this because this is – I mean, it is cool, all right? But the point is it's – I want to point out a couple of things. Bits and pieces of it are trivial, but one-half of it is not. But want to see people look at that and go like, "Wow, I need to really go to a quiet place, and sit, and actually think about what that means." That's pretty serious stuff, but I want to show you one thing. I hope this is gonna work out. We're gonna try.

I'm gonna at least – why don't I show you this part? Range of A is orthogonal to the null space of A transpose, right? That's one-half to the assertion. That's why the little perp is sitting above the plus, that's what you asked. So let's see. Let's show that. What do you have to show here? Well, really, I have to show that any element of this thing is orthogonal to any element of this one. I hope this works, by the way, but considering that we just posted these things online, it's almost certain I'm gonna end in a deep, deep hole, but let's just see what happens, all right?

So a general element of the range of A is a vector of the form, let's say, AX, and then I'll put a question mark here because this is what we have to show. Let's call a general element of null space of A transpose, I'm gonna call it W, but we put a little note, which is that A transpose W = 0. Oh, yeah. It's gonna work, okay? So this is what we, kind of, have to show here, and let's see. Well, I mean, to check orthogonality, you write this. The question mark means it's open; it's not shown yet. Okay.

Well, that's X transpose, A transpose W, and if you don't mind, I'm gonna just reassociate right now and skip a step, and I'm gonna look at that and put a question mark there, and I can erase the question mark because A transpose W is zero, okay? So now, the way you do that – this is the way you derive it. The way you hand in your homework or make your argument is the other way around.

You start and you say well, let's take W to be in the null space, that means this, and then you say let's let – here's the general element of range of A, it's AX. Then you say, well, let's just form the inner product this way, and you say let's start by writing this down. You first say to someone, "You'd agree with that, wouldn't you?" And they'd say, "Yes." And you go, "Well, no problem, but A transpose W is zero, right?" "Yes." "So I can then write this. You're not putting the question marks there." And they'd say, "I guess." And then you go, "Cool, then I can reassociate it and rewrite that that way." And they'd say, "Yes." And then you'd say, "Oh, I'm done." So that's the way that works. Anyway, what I wanted to point out is that in all of these things, some of them are very easy, and the others are not that easy. Okay. All right.

So I will start saying a few things about the next topic because it's the first one that is actually real and useful. It's least squares. I'll say a little bit about it, maybe just now because we're just finishing up, I'll just say a few things about it. It's not a new topic. It goes back to Gause who wrote, actually, a book about it and may or not have been the inventor of it, but as far as most people know, he was or something like that, and he wrote a book about it in Latin, which was actually just translated, which is, kind of, cool, and you can get the book. I mean, not that you're required to do so, but you get the book, and then on the left is the Latin, and on the right somebody translated it, so you get to see it.

And, by the way, if you read the book, you'll be very, very grateful you were born in a time when we actually, A.) Had notation because imagine what happens when you have to write out all of this stuff. Y = AX, that's, like, a page of Latin, okay? The transpose goes on, and on, and on. I mean, this is not simple stuff. That's No. 1, so that's the first thing you can be grateful for. Also, that you don't have to read it and the lecture notes for the class are not in Latin, another thing you could be grateful for.

The other one, actually, is that you were born in a time when computers will actually do all this stuff for you, and you can actually use this stuff, and embed it, and things like that. So anyway. All right, that's just history. I'm just telling you that least squares is not exactly a new topic. Okay. But I'll tell you what it's about. It's about approximate solution of over-determined equations. So let me say just a little bit about that so that Thursday we can do this.

Let's suppose you have Y = AX where A is skinny, okay? So that means something like you have more equations than unknowns. It depends on the context of what exactly it means. You have more specifications than you have designed degrees of freedom, lots of ways to say it. You have more measurements than things you have to, you know, it's a redundant measurement system, or at least, I should say, it's a dimensionally redundant

measurement system. It would appear, based on the sizes that you have more measurements than you need.

So people call this an over-determined set of linear equations, and basically for most, Y you can't solve that equation, obviously, because the range of A is a subspace of dimension M in RN. That's very small. In fact, even if M is N minus 1, in some, kind of, uniform distribution – with any probability density, if you draw Y from it, the probability that you can solve Y = AX is zero, right? Because a subspace, any strict subspace will have probability zero under any probability distribution, if you know about that kind of stuff. Okay.

So there's a compromise, and the idea is to not solve it but to approximately solve it. So you approximately – now, what does it mean to approximately solve? It means, instead of finding an X for which AX = Y, which is impossible, we're gonna find an X for which AX is approximately Y, and we'll measure proximity by the norm of the error. So that's the basic idea, and I think we will quit here.

[End of Audio]

Duration: 77 minutes