MIT OpenCourseWare
http://ocw.mit.edu

18.085 Computational Science and Engineering I, Fall 2008
Transcript – Lecture 14

PROFESSOR STRANG: OK. So this is Lecture 14, it's also the lecture before the exam on Tuesday evening. I thought I would just go ahead and tell you what questions there are, so you could see. I haven't filled in all the numbers, but this will tell you, it's a way of reviewing of course, to sort of see the things that we've done. Which is quite a bit, really. And also of course topics that will not be in the exam. You can see that they're not in the exam, for example. Topics from 1.7 on condition number, or 2.3 on Gram-Schmidt that I can speak about a little bit. So those are the, only thing I didn't fill in is the n time there. So I don't usually ask four questions, three is more typical. So it's a little bit longer but it's not a difficult exam. And I never want time to be the essential ingredient. So 7:30 to 9 is the nominal time, but it would not be a surprise if you're still going on after 9 o'clock a little bit. And I'll try not to, like, tear a paper away from you. So, just figure that if you move along at a reasonable speed, and so you can bring papers, the book, anything with you. And I'm open for questions now about the exam. You know where 54-100 is, it's a classroom unfortunately. Not a, sometimes we'll have the top of Walker where you have a whole table to work on, that's this 54-100 in the tallest building of MIT out in the middle of the middle space out there, is a large classroom. So if you kind of spread out your papers we'll be OK. It's a pretty big room. Questions. And, of course, don't forget this afternoon in 1-190 rather than here. So, review is in 1-190, 4 to 5 today. But if you're not free at that time don't feel that you've missed anything essential. I thought this would be the right way to tell you what the exam is and guide your preparation for it. Questions.

Yes, thanks.

AUDIENCE: [INAUDIBLE].

PROFESSOR STRANG: So these are four different questions. So this would be a question about getting to this matrix and what it's about, A transpose C A.

AUDIENCE: [INAUDIBLE]

PROFESSOR STRANG: OK, I don't use beam bending for that. I'm thinking of elastic bar. Yeah, stretching equation, yeah. So the stretching equation, so the first one we ever saw, the stretching equation, was just u'', or -u''=1. And now that allows a C to sneak in and that allows a matrix C to sneak in there, but I think you'll see what they should look like, but yeah.

AUDIENCE: [INAUDIBLE]

PROFESSOR STRANG: Yeah, yeah. So this is Section 2.2, directly out of the book. M will be the mass matrix, K will be the stiffness matrix, yep.

AUDIENCE: [INAUDIBLE]

PROFESSOR STRANG: OK. So, good point to say. In the very first section, 1.1, we gave the name K to a very special matrix, a specific one. But then later, now, I'm using the same letter K for matrices of that type. That was the most special, simplest completely understood case, but now I'll use K for stiffness matrices and when we're doing finite elements in a few weeks again it'll be K. Yeah, same name, right. So here you'll want to create K and M and know how to deal with, this was our only time dependent thing. So I guess what you're seeing here is not only what time dependent equation will be there but also that I'm not going in detail into those trapezoidal difference methods. Important as they are, we can't do everything on the quiz so I'm really focusing on things that were central to our course. Good. Other questions. I'm very open for more questions this afternoon. Yep.

AUDIENCE: [INAUDIBLE]

PROFESSOR STRANG: Others. OK. So, let me, I don't want to go on and do new material, because we're focused on these things. And this course, the name of this course is computational science and engineering. And by the way I just had an email last week from the dean of engineering, or a bunch of us did, to say that the school of engineering is establishing a center for computational engineering, CCE. Several faculty members there and, like myself in the School of Science and the Sloan School are involved with computation, and this new center is going to organize that. So, it's a good development. And it's headed by people in Course 2 and Course 16. So.

If we're talking about computations, and I do have to say something about how you would actually do the computations. And what are the issues about accuracy. Speed and accuracy is what you're aiming for in the computations. Of course, the first step is to know what problem is that you want to compute. What do you want to solve, what's the equation? That's what we've been doing all along. Now, I just take a little time out to say, suppose I have the equation. When I write K, I'm thinking of a symmetric positive, definite or at least semi definite matrix. When I write A I'm thinking of any general, usually tall, thin, matrix. Rectangular. So that I would need the squares for this guy, where straightforward elimination would work for that one. And so my first question is let's take this, so these are two topics for today. This one would come out of 1.7, that discussion with condition number. This one would come out of 2.3, the least squares section. OK, so if I give you, and I'm thinking that the computational questions emerge when the systems are large. So I'm thinking thousands of unknowns here. Thousands of equations, at the least. OK. So, and the question is, I do Gaussian elimination here, ordinary elimination. Backslash. And how accurate is the answer? And how do you understand? I mean, the accuracy of the answer is going to kind of depend on two things. And it's good to separate them.

One is the method you use, like elimination. Whatever adjustments you might make. Pivoting. Exchanging rows to get larger pivots. All that is in the algorithm, in the code. And then the second, very important aspect, is the matrix K itself. Is this a tough problem to solve whatever method you're using, or is it a simple problem? Is the problem ill condition, meeting K would be like nearly singular, and then we would know we had a tougher problem to solve whatever method, or is K quite well conditioned, I mean the best condition would be when all the columns are unit

vectors and all orthogonal to each other. Yeah, I mean that would be the best conditioning of all. That condition number would be one if this K, which, not too likely is a matrix that I would call Q. Q, which is going to show up over here, in the second problem is, Q stands for a matrix which has orthonormal columns. So, you remember what orthonormal means. Ortho is telling us perpendicular, that's the key point. Normal is telling us that they're unit vectors, lengths one. So that's the Q, and then you might ask what's the R. And the R is upper triangular. OK. OK.

So what I said about this problem, that there's the method you use, and also the sensitivity, the difficulty of the problem in the first place, applies just the same here. There's the method you use, do you use A transpose A to find u hat, this is, now we're looking for u hat, of course, the best solution. Do I use A transpose A? Well, you would say of course, what else. That equation, that least squares equation has A transpose Au hat equal A transpose b, what's the choice? But, if you're interested in high accuracy, and stability, numerical stability, maybe you don't go to A transpose A. Going to A transpose A kind of squares the condition number. You get to an A transpose A, that'll be our ok, but its condition number will somehow be squared and if the problem is nice, you're OK with that. But if the problem is delicate?

Now, what does delicate mean for Au=b? I'm kind of giving you a overview of the two problems before I start on this one. And then that one. So with this one the problem was, is the matrix nearly singular. What does that mean? Does MATLAB tell you, what does MATLAB tell you about the matrix? And that is measured by the condition number. What's the issue here is, when would this be a numerically difficult, sensitive problem? Well, the columns of A are not orthonormal. If they are, then you're golden. If the columns of A are orthonormal, then you're all set. So what's the opposite? Well, the extreme opposite would be when the columns of A are dependent. If the columns of A are linearly dependent and some column is a combination of other columns, you're in trouble right away. So that's like big trouble, that's like K being singular. Those are the cases. K singular here, dependent columns here. Not full rank. So, again we're supposing we're not facing disaster. Just near disaster. So we want to know, is K near the singular and how to measure that, and we want to know what to do when the columns of A are independent but maybe not very. And that would show up in a large condition number for A transpose A. And this happens all the time; if you don't set up your problem well, your experimental problem, you can easily get matrices A, whose columns are not very independent. Measured by A transpose A being close to singular. Right, everybody here's got that idea. If the columns of A are independent, A transpose A is non-singular. Fact, positive, definite.

Then now we're talking about when we have that property, but the columns of A are not very independent and the matrix A transpose A is not very invertible. OK, so that's what the things are. And then just to, because, say on this one, what's the good thing to do? the good thing to do is to call the qr code which gets its name because it takes the matrix A and it factors it. Of course, we all know that lu is the code here. It factors K. And qr is the code that factors A into a very good guy. An optimal queue. It couldn't be b. and an R, that's upper triangular, and therefore in the simplest form you see exactly what you're dealing with. Let me continue this least squares idea. Because Q and R are probably not so familiar. Maybe the name Gram-Schmidt is familiar? How many have seen Gram-Schmidt? The Gram-Schmidt idea I'll describe quickly, but just do those words, do those guys' names mean anything to? Yes, if they do. Quite a few. But not all. OK. OK. And I can I just say,

also Gram-Schmidt is kind of our name for getting these two factors. And you'll see why, it's very cool to have, why this is a good first step.

It costs a little to take that step, but if you're interested in safety, take it. It might cost twice as much as solving the A transpose A equation, so you double the cost by going this safer route. And double is not a big deal, usually. OK. So, I was going to say that Gram-Schmidt, that's the name everybody uses. But actually their method is no longer the winner. And in Section 2.3 and I'll try to describe a slightly better method than the Gram-Schmidt idea to arrive at Q and R. But let's suppose you got to Q and R. Then, what would be the least squares equation? A transpose Au hat is A transpose b, right? That's the equation everybody knows. But now if we have A factored into Q times R, let me see how that simplifies. So now A is QR, and A transpose, of course, is R transpose Q transpose u hat, and this is R transpose Q transpose b. Same equation, I'm just supposing that I've got a into this nice form where Q has, where I've taken these columns that possibly lined up too close to each other like, you know, angles of one degree. And I've got better angles. I've got these columns of A are too close. So I spread them out, two columns of qQ that are at 90 degrees. Orthogonal columns.

Now, what's the deal with orthogonal columns? Let me just remember the main point about Q. It has orthogonal columns, right, and I'll call those q's. q_1 to q_n. OK. And the good deal is what happens when I do Q transpose Q. So I do q_1 transpose, these now rows to qn transpose. q_1 columns to q_n. And what do I get when I multiply those matrices? Q transpose times Q. I get I. q_1 transpose q_1, that's the length of q_1 squared is one, and q_1 is orthogonal to all the others. And then q_2, you see I get the I. q_3, get an n by n. I get the identity matrix. Q transpose Q is I. That's the beautiful, just remember that fact. And use it right away. You see where it's used, Q transpose Q is I in the middle of that. So I can just delete that, I just have R transpose R and I can even simplify this further. What can I do now? So that's the identity, so I have R transpose R, but now I have an R transpose over here so am I left with? I'll multiply both sides by R transpose inverse and that will lead me to, Ru hat equals, knocking out our transpose inverse on both sides. Q transpose b. Well, that's, our least squares equation has become completely easy to solve. We've got a triangular matrix here, I mean it's just back substitution. It's just back substitution now, and a Q transpose b over there. So a very simple solution for our equation after the initial work of A=QR. OK. But very safe, Q is a great matrix to work with. In fact people, codes are written so as to use orthogonal matrices Q as often as they can.

Alright, so you had a look ahead of the computational side of 2.3, let me come back to the most basic equations just symmetric, positive, definite equations. Ku=f, and consider OK, how do we measure whether K is nearly singular. OK, let me just ask that question. That's the central question. How to measure, when K, which is, we're assuming be symmetric positive definite, nearly singular? How to measure that? How to know whether we're in any danger or not? OK. Well, first you might think OK, if it is singular its determinate is zero. So why not take its determinant? Well determinants, as we've said, are not a good idea numerically. First, they're not fun to compute. Second, they depend on the number of unknowns, right? If I just have twice the identity, suppose k is twice the identity matrix. You could not get a better problem than that, right? If k was twice the identity matrix the whole thing's simple. Or if K is, suppose K is one millionth of the identity matrix. OK, again, that's a perfect problem, right? If K is one millionth of the identity matrix, well to solve the problem you just multiply by a million, you've got the answer. So those are good.

And we have to have some measure of bad or good that tells us those are good. OK. So the determinant won't do. Because the determinant of two I would be two to the nth, the size of the matrix. Or the determinant of one millionth identity would be one millionth to the n. Those are not numbers we want. What's a better number? Maybe you could suggest a better number to measure how close is the matrix to being singular. What would you say?

I think if you think about it a little, so what numbers do we know? Well eigenvalues jumps to mind. Eigenvalues jumps to mind. Because this matrix K, being symmetric positive definite, has eigenvalues say lambda_1 less than lambda_2, so on. So on. Up to, so this is lambda_max and that's lambda_min, and they're all positive. And so what's your idea of whether the thing's nearly singular now? Look at lambda_1, right? If lambda_1 is near zero, that somehow indicates near singular. So lambda_1 is sort of a natural test. Not that I intend to compute lambda_1, that would take longer than solving the system. But an estimate of lambda_1 would be enough. OK. But my answer is not just lambda_1. And why is that? Because the examples I gave you, when I had twice the identity, what would lambda_1 be there in that case? If my matrix K was beautiful, twice the identity matrix, lambda_1 would be two. All the eigenvalues are two for the identity matrix. Now if my matrix was one millionth of the identity, again I have a beautiful problem. Just as good, just as beautiful problem. What's lambda_1 for that one? One millionth. It looks not as good, it looks much more singular, but that's not really there. So you could say, we'll scale your matrix. And scaling the matrices, in fact scaling individual rows and columns to get it, you might have used, your unknowns might be somehow in the wrong units. So one of the answers is way big and the second component is way small. That's not good. So scaling is important. But even then you still end up with a matrix K, some eigenvalues and I'll tell you the condition number.

The condition number of K is the ratio of this guy to this one. In other words, two K, or a million K, or one millionth K, all have the same condition number. Because those problems are identical problems. Multiplying by two, multiplying by a million, dividing by a million didn't change reality there. So if we're in floating points, it just didn't change. So the condition number is going to be lambda_max/lambda_min. And this is for symmetric positive definite matrices. And and MATLAB will print out that number. Or print an estimate for that number; as I said we don't want to compute it exactly. Lambda_max/lambda_min. That measures how sensitive, how tough your problem is. OK. And then I have to think how does that come in, why is that an appropriate number? I guess I've tried to give an instinct for why it's appropriate, but we can be pretty specific about it. In fact, let's do that now.

So what would be the condition number of twice the identity? It would be one. Perfectly conditioned problem. What would be the condition, yeah, OK. What would be the condition of a diagonal matrix suppose K was a diagonal matrix two, three, four? The condition number of that matrix is two, right? Lambda_max is sitting there, lambda_min is sitting there, the ratio is two. Of course, any condition number under 100 or 1000 is no problem. Roughly the rule of thumb is that the, what's the rule of thumb? I think that maybe the number of digits in the condition number, the number of digits, maybe if the condition number was 1000 you would be taking a chance that your last three digits, single precision that would be three out of six, three bits, somehow the log of the condition number, the number of digits in it, is some measure of the number of digits you'd lose. Because you're doing floating point, of course, here. That's it, totally well conditioned matrix. I wouldn't touch that one. I mean that's just fine. But we can figure out, here's the point that I should

make. Because here's the computational science point. When this is our special K, our -1, 2, -1 matrix. -1, 2, -1 matrix of size n, the condition number goes, like, n squared. Because we know the eigenvalues of that matrix, we could see it. The largest eigenvalue, when n is big, say n is 1000. And we're dealing with our standard second difference matrix, the most important example I could possibly present. Then the largest eigenvalues we actually are there in Section 1.5, we didn't do them in detail, we'll probably come back to them when we need them. But the largest one is about four. And the smallest one is pretty small. The smallest one is sort of like a sine squared of a small number. And so the smallest eigenvalue is of order 1/n squared.

And then when I do that ratio of four, lambda_max is just like four, this lambda_min is like 1/n squared, quite small. That ratio gives me the n squared. So there's an indication. Basically, that's not bad. That's not bad if n is 1000 in most engineering problems, that gives you extremely, extremely good accuracy. Condition number of a million, you could live with. If n is 100 more typical, condition number of 10,000 is basically, I think OK. And I would go with it. But if the condition number is way up, then I'd think again, did I model the problem well. OK. Alright. So that's, now I have to tell you why is this inappropriate? How do you look at the error? So can I write down a way of approaching this does Ku=f? So this is the first time I've used the word round off error. So in all the calculations, in all the calculations that you have to do, to get to u, those row operations and you're doing them to the right side too, so all those are floating point operations in which small errors are sneaking in. And it was very unclear, in the early years, whether the millions and millions of operations that you do addition, subtractions, multiplications, and elimination, do those, could those add up, if they don't cancel, you've got problems, right? But in general you would expect somehow that these are just round off errors, you're making them millions and millions of times, it would be pretty bad luck, I mean like Red Sox twelfth inning bad luck to have them pile up on you. So you don't expect that.

Now, what you do solve, so what you actually compute, so this is the exact. This it would be the computed. Let me suppose that the computed one is sort of a, there's an error. I'll call it delta u. That's our error. And it's equal to an f plus delta f. And this is our round off error, this is error we make, and this is error in the answer. In the final answer. OK, now I would like to have an error equation. An equation for that error, delta u, because that's what I'm trying to get an idea of. No problem. If I subtract the exact equation from this equation, I have a simple error equation. So this is my error equation. OK. So I want to estimate the size of that error, compared to the exact. You might say, and you would be right, in saying, well wait a minute as you do all these operations you're also creating errors in K. So I could have a K plus delta K here, too. And actually it wouldn't be difficult to deal with. And would certainly be there in a proper error analysis. And it wouldn't make a big difference, the condition number would still be the right measure.

So let me concentrate here on the error in f, when subtracting one from the other gives me this simple error equation. So my question is, when is that error, delta u, big? When do I get a large error? And delta f, I'm not controlling. I might control the size, but the details of it I can't know. So what delta f, now I'll take worst possible here. Suppose this is of some small size, ten to the minus something, times some vector of errors, but I don't know anything about that vector and therefore I'd better take the worst possibility. What would be the worst possibility? What right hand side would give me the biggest delta u? Yeah. Maybe that's the right question to ask. So now we're being a little pessimistic. We're saying what right hand side, what set of

errors in the measurements or from the calculations would give me the largest delta u? Well, so let's see. I'm thinking the worst case would be if delta f was an eigenvector with the smallest eigenvalue, right? If delta f is an eigenvector, is x_1, the eigenvector that goes with lambda_1, the worst case would be for that to be the first eigenvector. That would be the worst direction. Of course, it would be multiplied by some little number. Epsilon is every mathematician's idea of a little number. OK. So epsilon x_1, then what is delta u?

Then the worst delta u is what? What would be the solution to that equation? If the right hand side was epsilon and an eigenvector? This is the whole point of eigenvectors. You can tell me what the solution is. Is it a multiple of that eigenvector? You bet. If this is an eigenvector, then I can put in the same eigenvector there, I just have to scale it properly. So it'll be just this side and what do I need? I think I just need lambda_1. The worst K inverse can be is like 1/lambda_1. Right, if I claim that that's the answer, if the right hand side is sort of in the worst direction, then the answer is that same right hand side divided by lambda_1. Let me just check. If I multiply both sides by K, I have K delta u equals K, what's K*x_1? Everybody with me? What's K*x_1? Lambda_1*x_1, so the lambda_1's cancel, then I got the epsilon x_1 I want. So, no surprise. That's just telling us that the worst error is an error in the direction of the low eigenvector, and that error gets amplified by 1/lambda_1. OK. So that's brought lambda_1, lambda_min into it, in the denominator.

Now, here's another point. Second point now. So that would be the absolute error. But we saw for those factors of two and one millionth and so on, that really it's the relative error. So I want to estimate not the absolute error, delta u, but the error delta u relative to u itself. So that if I scale the whole problem, my relative error wouldn't change. So, in other words, what I want to do is ask in this case, what's the right hand side? How big, yeah, I know, I want to know how small u could be, right? I'm shooting for the worst. The relative error is the size of u, maybe I should put it this way. The relative error is the size of the error relative to the size of u. And I want to know how big that could be. OK. So now I know how big delta u could be, it could be that big. But u itself, how big could u be? How small could u be, right? u's in the denominator. So if I'm trying to make this big, I'll try to make that small. So when is u the smallest? Over there I said when is delta u the biggest, now I'm going to say when is u the smallest? What f would point me in the direction in which u was the smallest? Got to be the other eigenvector. This end. The worst case would be when this is in the direction of x_n. The top eigenvector. In that case, what is u?

So I'm saying the worst f is the one that makes u smallest, and the worst delta f is the one that makes delta u biggest. I'm going for the worst case here, so if the right side is x_n, what is u? x_n/lambda_n. Because when I multiply by K, K times x_n brings me a lambda_n, cancel that lambda_n I get it right. So there is the smallest u, and here is the largest delta u. And the epsilon is coming from the method we use, so that's not involved with the matrix K. So do you see on there, if I'm trying to estimate delta u over u, that's b. The size of delta to u over this, the size of delta u over the size of u is what? Delta u has some epsilon that measures the machine number of digits we're keeping, the machine length, word length and so on. This is a unit vector over lambda_1. That's delta u over lambda_1, this u is in the denominator. lambda_n u is down here, so the lambda_n flips up. Do you see it? By taking the worst case, I've got the worst relative error. So that for other methods, other f's and delta f's, they won't be the very worst ones. But here I've written down what's the worst. And that's the reason that this is the condition number. So I'm

speaking about topics that are there in 1.7, trying to give you the main point. The main point is, look at relative error because that's the right thing to look at. Look at the worst cases, which are in the directions of the top and bottom eigenvectors. In that case, that relative error has this condition number, lambda_n/lambda_1, and that's the good measure for how singular the matrix is. So one millionth of the identity is not a nearly singular matrix, because lambda_max and lambda_min are equal, that's a perfectly conditioned matrix.

This matrix has condition number two, 4/2. It's quite good. This matrix is getting worse with an n squared in there; if n is big and other matrices could be worse than that. OK. so that's my discussion of condition numbers. I'll add one more thing. These eigenvalues were a good measure when my matrix was symmetric positive definite. If I have a matrix that I would never call K, a matrix like one, one, zero and a million, OK, that I would never write K for that. I would shoot myself first before writing K. So what are the eigenvalues of lambda_min lambda_max for that matrix? Are you up now on eigenvalues? We haven't done a lot of eigenvalues but triangular matrices are really easy. The eigenvalues of that matrix are? One and one. The condition number should not be 1/1, that would be bad. So instead, if this was my matrix and I wanted to know its condition number, what would I do? How would I define the condition number of A? You know what I do whenever I have a matrix that is not symmetric. I get to a symmetric matrix by forming A transpose A, I get a K, I take its condition number by my formula, which I like in a symmetric case, and then I would take the square root.

So that would be a pretty big number. For this, for that matrix A, the condition number of that matrix A is up around 10^(6). The condition number of that matrix is up around 10^(6) even though it's eigenvalues are one and one because when I form A transpose A, those eigenvalues will jump all over. And then probably this thing will have eigenvalues way up and the condition number will be high. OK. So that's a little, you'll meet condition number. MATLAB shows it, and you naturally wonder what is this? Well, if it's a positive definite matrix, it's just the ratio lambda_max to lambda_min and it tells you as it gets bigger that means the matrix is tougher to work with. OK. We have just five minutes left to say a few words about QR. OK, can I do that in just a few minutes? And much more is in the codes.

OK, what's the deal with QR? I'm starting with a matrix A. Let's make it two by two, a two by two. OK, it's got a couple of columns. Can I draw them? So it's got a column there, that's its first column and it's got another column there, maybe that's not a very well conditioned matrix. Those are the columns of a. Plotted in the plane, two space. OK, so now the Gram-Schmidt idea is out of those columns, get forced to normal columns. Get from A to Q. So the Gram-Schmidt idea is, out of these two vectors, two axes that are not at 90 degrees produce vectors that are at 90 degrees. Actually, you can guess how you're going to do it. Let me say, OK I'll settle for that direction, that can be my first direction, q_1. What should be q_2? If that direction is the right one for q_1 I say OK I'll settle for that, what's the q_2 guy? Well, what am I going to do? I mean, Gram thought of it and Schmidt thought of it. Schmidt was a little later, but it wasn't, like that hard to think of it. What do you do here? Well, we know how to do projections from this least squares. What am I looking for? Subtract off the projection, right. Take the projection and subtract it off and be left with the component that's perpendicular. So this will be the q_1 direction and this will be, this guy e, that what we call e would tell me the q_2 direction. And then we could make those into unit vectors and we'd be golden.

And if we did that we would discover that the original a_1, a_2 and a_1, so this is the original first column, and the original second column, would be the good q_1 and the good q_2 times some matrix R. So here's our A=QR. It's your chance to see this second major factorization of linear algebra. LU being the first, QR being the second. So what's up? Well, compare first columns. First columns, I didn't change direction. So all I have is here is some scaling r_1,1, zero. Some number times q_1 is a_1. That direction was fine. The second direction, q_2 and a_2, those involve also a_1. So there's an r_1,2 and an r_2,2 there. The point was, this came out triangular. And that's what makes things good. It came out triangular because of the order that Gram and Schmidt worked. Gram and Schmidt settled the first one in the first direction. Then they settled the first two in the first two directions. If we were in three dimensions, there'd be an a_3 somewhere here, coming out of the board. And then the q_3 would come straight out of the board. Right? If you just see that you've got Gram-Schmidt completely. a_1 is there. So is q_1. a_2 is there. I'm in the board still, in the plane of a_1 and a_2, is the plane of q_1 and q_2, I'm just getting right angles in. a_3, the third column in a three by three case, is coming out at some angle. I want q_3 to come out at a 90 degree angle. So that q_3 will involve some combination of all the a's. So if it was three by three, this would grow to q_1, q_2, q_3, and this would then have three guys in its third column. But maybe you see that picture.

So that's what Gram-Schmidt achieves. And I just can't let time right now without saying that this is a pretty good way. Actually, nobody's thought there was a better one for centuries. But then a guy named Householder came up with a different way, and a numerically little better way. Numerically a little better way. So this is the Gram-Schmidt way. Can I just put those words up here? So there's the Gram-Schmidt, the classical Gram-Schmidt idea, which was what I described, was the easy one, easy to describe. And then there's a method called Householder, just named after him, MATLAB would follow. That every good qr code now uses Householder matrices. It achieves the same results. And if I had a little bit more time I could draw a picture of what it does. But there you go. So that's my lecture on, my quick lecture on numerical linear algebra. These two essential points and I'll see you this afternoon. Let me bring those quiz questions down again, for any discussion about the quiz.