PROFESSOR STRANG: OK. Well, we had an election since I saw you last. I hope you're happy about the results. I'm very happy. Except one thing. There was a senator who was in 18.085. And he lost his seat. So we don't have a senator from 18.085 any more. It was Senator Sununu, from New Hampshire. So he took 18.085. Actually, it was a funny experience. So, some years ago I had to go to Washington to testify before a committee about increasing the funding for mathematics. So I was nervous, of course, trying to remember what I'm going to say. More money is the main point, but you have to say it more delicately than that. And so I got in, a lot of people were taking turns, you only get five minutes to do it. And so I went and sat down, waited for my turn. And here on the committee was my student, Senator Sununu. Well at that time, I think, Representative Sununu. So that was nice. He's a nice guy. Actually, the lobbyist who kind of guided me there, and found the room for me and sort of told me what to do, said I hope you gave him an A. And the truth is I didn't. So I guess you guys should all get As, just to be on the safe side. Or let me know if you're going to be in the Senate anyway. So there you go. Anyway. So alright. But he wasn't great in 18.085. OK, so you know what, this evening-- I just called the schedules office to confirm that it is tonight, in 54-100. It's not a long, difficult exam so see you at 7:30, or see you earlier at 4:00 for a review of everything that you want to ask about. So I won't try to do an exam review this morning; we've got lots to do and we're into November already.

So I want to finish the discussion of the fast Poisson solver. Because it's just a neat idea which is so simple. Anytime you have a nice square grid. I guess the message is, anytime you have a nice square or rectangular grid, you should be thinking, use the fast Fourier transform somehow. And here it turns out that the eigenvectors are, for this problem, this K2D, so we're getting this matrix K2D. Which is of size N squared by N squared, so a large matrix. But its eigenvectors will be, well, actually they'll be sine functions for the K2D, the fixed-fixed. And they would be cosine functions for the B2D, free-free. And the point is the fast Fourier transform does all those fast. I mean, the fast Fourier transform is set up for the complex exponentials as we'll see in a couple of weeks. That's the most important algorithm I could tell you about. After maybe Gaussian elimination, I don't know. But my point here is that if you can do complex exponentials fast, you can do sines fast, you can do cosines fast, and the result will be that instead of-- I did an operation count when I did elimination in the order one, two, three, four, five, six, ordered by along each row. That gave us a matrix that I want to look at again, this K2D matrix. If I did elimination as it stands, I think the count was N^4. I think the count was, size N squared, I had N squared columns to work on and I think each column took N squared steps, so I was up to N^4. And that's manageable. 2-D problems, you really can use backslash. But for this particular problem, using the FFT way, it comes down to N squared log N. So that's way better. Way better. Big saving. In 1-D, you don't see a saving. But in 2-D it's a big saving, and in 3-D an enormous saving. So a lot of people would like to try to use the FFT also when the region isn't a square. You can imagine all the thinking that goes into this. But we'll focus on the one where it's a square. Oh, one thing more to add. I spoke about reordering the unknowns. But you probably were wondering, OK, what's that about because I only just referred to it. Let me suggest an ordering and you can tell me what the matrix K2D will look like in this ordering. I'll call it the red-black ordering. I guess with the election just behind us I could call it the red-blue ordering. OK, red-blue ordering, OK. So the ordering, instead of ordering it by-- Let me draw this again, I'll put in a couple more, make N=4 here, so now I've got 16, N squared is now 16. So here's the idea. The red-blue ordering is like a checkerboard. The checkerboard ordering, you could think of. This will be one, this'll be two, this'll be three, this'll be four, five, six, seven, eight. So I've numbered from one to eight. And now 9 to 16 will be the other guys. These will be 9, 10, 11, 12, 13, 14, 15, 16.

So suppose I do that, what K2D look like? Maybe you could, it's a neat example. It gives us one more chance to look at this matrix. So K2D is always going to have fours on the diagonal. So I have 16 fours. Whatever order I take, the equation at a typical point, like say this one, that's point number one, two. That's point number three, at a point number three I have my five point molecule with a four in the middle. So it'd be a four in this, on that position. And then I have the four guys around it, the minus ones, and where will they appear in the matrix? With this ordering. So you get to see the point of an ordering. So this was point one, point two, point three, I'm focusing on point three. It's got a four on the diagonal and then it's got a minus one, a minus one, a minus one, a minus one, where do they appear? Well, all those guys are what color? They're all color blue. Right? All the neighbors are colored blue. So I won't get to them until I've gone through the red, the eight reds, and then I come back to nine. Whatever, 11, 12, 13. The point is they'll all be over here. So this is like, you see that the only nodes that are next to a red node are all blue. And the only nodes next to a blue node are red. So this is filled in a little bit, that's where I've sort of like four diagonals of minus ones or something. But there you go. That gives you an idea of what a different ordering could be. And you see what will happen now with elimination? Elimination is completed already on this stuff. So all the-- With elimination you want to push the non-zeroes way to the end. That's sort of like the central idea is, it's kind of a greedy approach. Greedy algorithm. Use up zeroes while the sun's shining. And then near the end, where the problem is, elimination's reduced it to a much smaller problem, then you've got some work to do. So that's what would happen here.

By the way there's a movie, on the 18.086 page. I hope some of you guys will consider taking 18.086. It's a smaller class, people do projects like creating this elimination movie. So that would be math.mit.edu/18.086, and with the movie-- has a movie of different orderings. That to try to figure out what's the absolute best ordering with the absolute least fill-in is, that's one of these NP-hard combinatorial problems. But to get an ordering that's much better than by rows is not only doable but should be done. OK, so that's the ordering for elimination. As this is for elimination. OK. So that's a whole world of its own. Ordering the thing for Gaussian elimination. My focus in this section 3.5, so this is Section 3.5 of the book and then this is Section 3.6, which is our major one. That's our big deal. But my focus in 3.5 is first to try to see what the 2-D matrix is. I mean, a big part of this course, I think, a big part of what I can do for you, is to, like, get comfortable with matrices. Sort of see what do you look at when you see a matrix? What's its shape? What are its properties? And so this K2D is our first example of a 2-D matrix, and it's highly structured. The point is we'll have K2D matrices out of finite elements. But the finite elements might be, well, they could be those. This could be a finite element picture. And then the finite element matrix on such a regular mesh would be quite structured too. But if I cut them all up into triangles and I have a curved region and everything, an unstructured mesh, then I'll still have the good properties, this will still be symmetric positive definite, all the good stuff. But the FFT won't come in.

OK, so now I want to take a look again at this K2D matrix, OK? So one way to describe the K2D matrix is the way I did last time, to kind of write down the typical row, typical row in the middle. It's got a four on the diagonal and four minus ones. And here we've re-ordered it so the four minus ones came off. Came much later. But either way, that's what the matrix looks like. Let me go back to this ordering. And let's get 2-D, another better, clearer look at K2D. I want to construct K2D from K. Which is this K1D. I don't use that name, but here it is. Let me just show it to you. OK, so let me take the matrix that does the second differences in the x direction. This'll be an N squared by N squared matrix. And it'll take second differences on every row. Let me just write in what it'll be. It'll be K, it'll be a block matrix. K, K, n blocks. Each n by n. You see that that will be the second difference matrix? K takes the second differences along a row. This K will take the second x differences along the next row. Row by row, simple. So this is the u_xx part. Now what will the u_yy, the second y derivative-- That gives second differences up the columns, right? So can I see what that matrix will look like, second differences up the columns? Well, I think it will look like this. It will have twos on the diagonal. 2I, 2I, 2I, this is second differences, right? Down to 2I. And next to it will be a minus the identity-- Let me write it, and you see if you think it looks good. So minus the identity. It's like a blown up K. Somehow. Right? I have, do you see, it's every row has got a two and two minus ones. That's from the minus one the two and the minus one in the vertical second difference. But you see how the count, you see how the numbering changed it from here?

Here the neighbors were right next to each other, because we're ordering by rows. Here I have to wait a whole N to get the guy above. And I'm also N away from the guy below. So this is the two, minus one, minus one centered there, above and below. Do you see that? If you think about it a little it's not sort of difficult to see. And I guess the thing I want also to do, is to tell you that there's a neat little MATLAB command, or neat math idea really, and they just made a MATLAB command out of it, that produces this matrix and this one out of K. Can I tell you what this is? It's something you don't see in typical linear algebra courses. So I'm constructing K2D from K1D by, this is called a Kronecker product. It's named after a guy, some dead German. Or sometimes called tensor product. The point is, this is always the simplest thing to do in two dimensions or three dimensions, or so on. Is like product of 1-D things, like copy the 1-D idea both ways. That's all this thing is doing. Copying the one idea in the x direction and in the y direction. So the MATLAB command, I've got two pieces here. For this first piece it's kron, named after Kronecker, of-- Now, let's see. I'm going to put two N by N matrices, and Kronecker product is going to be a matrix, a giant matrix, of size N squared. Like these. OK, so I want to write the right thing in there. I think the right thing is the identity and K. OK, and so I have to explain what this tensor product, Kronecker product, is. And this guy happens to be also a Kronecker product. But it's K and I. So I'm just like mentioning here. That because if you have 2-D problems and 3-D problems and they're on a nice square grid so you can like just take products of things, this is what you want to know about. OK.

So what is this Kronecker product? Kronecker product says take the first matrix, I. It's N by N. Just N by N, that's the identity. And now take this K and multiply every one of those numbers, these are all numbers, let me put more numbers in, so I've got 16 numbers. This is going to work for this mesh that has four guys in each direction. And four directions. Four levels. So I is four by four, and K is four by four and now, each number here gets multiplied by K. So because of all those zeroes I get just K, K, K, K. Which is exactly what I wanted. So that Kronecker product just-- In one step you've created an N squared by N squared matrix that you really would not want to type in entry by entry. That would be a horrible idea. OK, and if I follow the same principle here, I'll take the Kronecker product here, as I start with a matrix K, so I start with two, minus one, minus one; two, minus one, minus one; two, minus one, minus one; two. That's my K, and now what's the Kronecker idea? Each of those numbers gets multiplied by this matrix. So that two becomes 2I, minus one becomes minus I, minus one becomes minus I, it all clicks. And it's producing exactly the u_yy part, the second part. So that's the good construction, just to tell you about. And the beauty is, this is just like cooked up, set up for separation of variables. If I want to know the eigenvalues and the eigenvectors of this thing, I start by knowing the eigenvalues and eigenvectors of K. Can you remind me what those are? We should remember them. So, anybody remember eigenvectors of K? Well, this is going back to Section 1.5, way early in the semester. And there's a lot of writing involved and I probably didn't do it all. But let me remind you of the idea. We guessed those eigenvectors. And how did we guess them? We guessed them by comparing this difference equation to the differential equation. So we're in 1-D now, I'm just reminding you of what we did a long time ago.

OK, so what did we do? I want to know the eigenvectors of these guys. So I better know the eigenvectors of the 1-D one first. OK, so what did we do in 1-D? We found the eigenvectors of K by looking first at the differential equation -u''=lambda*u with-- And remember we're talking about the fixed-fixed here. And anybody remember the eigenvectors of this guy, eigenfunctions I guess I should say for those? So sines and cosines look good here, right? Because you take their second derivative, it brings down the constant. And then do I want sines or cosines? Looking at the boundary conditions, that's going to tell me everything. I want sines, cosines are wiped out by this first condition that u(0) should be zero. And then the sine has to come back to zero at one, so the eigenfunctions were u equals the sine of something, I think. I want a multiple of pi, right? pi*k*x. I think that would be right. Because at x=1, that's come back to zero. And the eigenvalue lambda, if I just plug that in, two derivatives bring down pi*k twice. And with a minus, and that cancels that minus. So the eigenvalues were pi squared k squared. And that's the beautiful construction for differential equations. And these eigenfunctions are a complete set, it's all wonderful. So that's a model problem of what classical applied math does for Bessel's equation, Legendre's equation, a whole long list of things. Now, note all those equations that I just mentioned would have finite difference analogs. But to my knowledge, it's only this one, this simplest, best one of all, we could call it the Fourier equation if we needed a name for it. I just thought of that. That sounds good to me. Fourier equation. OK.

So what's great about this one is that the eigenvectors in the matrix case are just found by sampling these functions. You're right on the dot if you just sample these functions. So, for K, the eigenvectors, what do I call eigenvectors? Maybe y's, I think I sometimes call them y. So a typical eigenvector y would be a sine vector. I'm sampling it at, let's see, can I use h for this step size? So h is 1/(n+1). As we saw in the past. h is 1/(n+1). So a typical eigenvector would sample this thing at h, 2h, 2h, 2h, sin(2*pi*k*h), and so on, dot dot dot dot. And that would be an eigenvector. That would be the eigenvector number k, actually. And do you see that that sort of eigenvector is well set up. It's going to work because it ends, where does it end? Who's the last person, the last component of this eigenvector? It's sin(N*pi*k*h). Sorry about all the symbols, but compared to most eigenvectors this is, like, the greatest. OK, now why do I like that? Because the pattern, what would be the next one if there was an N plus first? If there was an N plus first component, what would it be? What would be the sine of (N+1)*pi*k*h? That's the golden question. What would be, I'm trying to say that these have a nice pattern because the guy that's-- the next person here would be sin((N+1)*pi*k*h), which is what? So N+1 is in there. And h is in there, so what do I get from those guys? The (N+1)h is just one, right? (N+1)h carries me all the way to the end. That's a guy that's out, it's not in our matrix because that's the part, that's a known one, that's a zero in a fixed boundary condition.

So like, the next guy would be sine (N+1)h, that's one. sin(pi*k). And what is sin(pi*k)? The sine of pi*k is always? Zero. So we're getting it right, like, the guy that got knocked off following this pattern will be zero. And what about the guy that was knocked off of that end? If this was sin(2pi*k*h), and sin(pi*k*h), this would be sin(0pi*k*h), which would be? Also zero, sin(0). So, anyway you could check just by, it takes a little patience with trig identities, if I multiply K by that vector, the pattern keeps going, the two minus one, minus one, you know at a typical point I'm going to have two of these, minus one of these, minus one of these. And it'll look good. And when I get to the end I'll have two minus one of these. The pattern will still hold, because the minus one of these I can say is there. But it is a zero, so it's OK. And similarly here. Anyway, the pattern's good and the eigenvalue is? Something. OK, it has some formula. It's not going to be exactly this guy. Because we're taking differences of sines and not derivatives of sines. So it has some formula. Tell me what you would know about lambda. I'm not going to ask you for the formula, I'm going to write it down. Tell me, before I write it down, or I'll write it down and then you can tell me. I have two on the diagonal, so that's just going to give me a two. And then the guy on the left and the guy on the right, two, two with minus ones I think gives two, I think it turns out to be a cosine of k-- a k has to be in there, a pi has to be in there, and h. I think that'll be it. I think that's the eigenvalue. That's the k-th eigenvalue to go with this eigenvector. What do you notice about two minus two times the cosine of something? What is it? Positive, negative, zero, what's up? Two minus to times a cosine is always? Positive. What does that tell us about K that we already knew? It's positive definite, right? All eigenvalues, here we actually know what they all are, they're all positive. I'm never going to get zero here. These k*pi*h's are not hitting the ones where the cosine is one and, of course, you can imagine. So I started this sentence and realized I should add another sentence. These are all positive. We expected that. We knew that K was a positive definite matrix. All its eigenvalues are positive, and now we actually know what they are.

And if I had the matrix B instead, for a free-free one? Then this formula would change a little bit. And what would be different about B, the free-free matrix? Its eigenvectors would be maybe at half angles or some darned thing happens. And so we get something slightly different here. Maybe h is 1/n for that, I've forgotten. And what's the deal with the free-free K? One of the eigenvalues is zero. The free-free is the positive semi-definite example. OK, that was like a quick review of stuff we did earlier. And so I'm coming back to that early point in the book, because of this great fact that my eigenvectors are sines. So the point is, my eigenvectors being sines, that just lights up a light saying use the fast Fourier transform. You've got a matrix full of sine vectors. Your eigenvector matrix is a sine transform. It's golden, so use it. So I'll just remember then how, recall that-- So what are the eigenvectors in 2-D? First of all, so let's go to 2-D. Let me do the continuous one first. Yeah, -u_xx-u_yy=lambda*u. The eigenvalue problem for Laplace's equation now in 2-D, and again I'm going to make it on the square. The unit square. And I'm going to have zero boundary conditions. Fixed-fixed. Anybody want to propose an eigenfunction for the 2-D problem? So the 2-D one, the whole idea is hey, this square is like a product of intervals somehow. We know the answer in each direction. What do you figure, what would be a good eigenfunction u(x,y)? Or eigenfunction, yeah, u(x,y). For this problem? What do you think? This is like the-- The older courses on applied math did this until you were blue in the face. Because there wasn't finite elements and good stuff at that time. It was exact formulas. You wondered about exact eigenfunctions and for this problem variables separate. And you get u(x) is the product of sine, of this guy, sin(pi*k*x), times the sine of pi*l*y.

Well, once you have the idea that it might look like that, you just plug it in to see, does it really work? And what's the eigenvalue? So I claim that that's a good eigenfunction function and I need, it's got two indices, k and l, two frequencies. It's got an x frequency and a y frequency. So I need double index. k and l will go, yeah, all the way out to infinity in the continuous problem. And what's the eigenvector? Can you plug this guy in? What happens when you plug that in? Take the second x derivative, what happens? A constant comes out and what's the constant? pi squared k squared. Then plug it into that term. A constant comes out again. What's that constant? pi squared l squared. They come out with a minus and then you already built in the minus, so that they've made it a plus. So the lambda_kl is just pi squared k squared, plus pi squared l squared. You see how it's going? If we knew it in 1-D now we get it in 2-D, practically for free. Just the idea of doing this. OK, and now I'm going to do it for finite differences. So now I have K2D and I want to ask you about its eigenvectors, and what do you think they are? Well, of course, they're just like those. They're just, the eigenvectors, shall I call them z_kl? So these will be the eigenvectors. The eigenvectors z_kl will be, their components-- I don't even know the best way to write them. The trouble is this matrix is of size N squared. Its eigenvectors have got N squared components, and what are they? Just you could tell me in words. What do you figure are going to be the components of the eigenvectors in K2D when you know them in 1-D? Products, of course. This construction, however I write it, is just like this. z_kl, a typical component, typical components-- So the components of the eigenvectors are products of these guys, like sin(k*pi*h), something like that. I've got indices that I don't want to get into. Just damn it, I'll just put that. Sine here or something, right. We could try to sort out the indices, the truth is a kron operation does it for us.

So all I'm saying is we've got an eigenvector with N components in the x direction. We've got another one with index l, N components in the y direction. Multiply these N guys by these N guys, you get N squared guys. Whatever order you write them in. And that's the eigenvector. And then the eigenvalue is, so the eigenvalue will be, well what do we got? We have a second x difference, so it will be a 2-2cos(k*pi*h). From the xx term. And it'll plus the other term will be a 2-2cos(l*pi*h). That's cool, yeah. I didn't get into writing the details of the eigenvector, that's a mess. This is not a mess. This is nice. What do I see again? I see four coming from the diagonal. I see two minus ones coming from left and right. I see two minus ones coming from up and below. And do I see a positive number? You bet. Right, this is positive, this is positive. Sum is positive. The sum of two positive definite matrices is positive definite. I know everything about K. And the fast Fourier transform makes all those calculations possible. So maybe I, just to conclude this subject, would be to remind you, how do you use the eigenvalues and eigenvectors in solving the equation? I guess I'd better put that on a board. But this is what we did last time. So the final step would be how do I solve (K2D)U=F? You remember, what were the three steps? I do really want you to learn those three steps. It's the way eigenvectors and eigenvalues are used. It's the whole point of finding them. The whole point of finding them is to split this problem into N squared little 1-D problems, where each eigenvector is just doing its own thing. So what do you do? You write F as a combination, with some coefficients c_kl, of the eigenvectors y_kl. So these are vectors. Put an arrow over them just to emphasize. Those are vectors. Then what's the answer, let's skip the middle step which was so easy. Tell me the answer.

Supposed the right-hand side is a combination of eigenvectors. Then the left-hand side, the answer, is also a combination of eigenvectors. And just tell me, what are the coefficients in that combination? This is like the whole idea is on two simple lines here. The coefficient there is? What do I have? c_kl, does that come in? Yes, because c_kl tells me how much of this eigenvector is here. But now, what else comes in? lambda_kl, and what do I do with that? Divide by it, because when I multiply that'll multiply by it and bring back F. So there you go. That's the-- This U is a vector, of course. These are all vectors. 2-D, we'll give them an arrow, just as a reminder that these are vectors with N squared components. They're giant vectors, but the point is this y_kl, its N squared components are just products, as we saw here, of the 1-D problem. So the fast Fourier transform, the 2-D fast Fourier transform just takes off. Takes off and gives you this fantastic speed. So that's absolutely the right way to solve the problem. And everybody sees that picture? Don't forget that picture. That's like a nice part of this subject, is to get-- It's formulas and not code. But it's easily turned into code. Because the FFT and the sine transform are all coded. Actually, Professor Johnson in the Math Department, he created the best FFT code there is. Do you know that? I'll just mention. His code is called FFTW, Fastest Fourier Transform in the West, and the point is it's set up to be fast on whatever computer, whatever architecture you're using. It figures out what that is. And optimizes the code for that. And gives you the answer. OK. That's Part 2 complete, of Section 3.5. I guess I'm hoping after we get through the quiz, the next natural step would be some MATLAB, right? We really should have some MATLAB case of doing this. I haven't thought of it, but I'll try. And some MATLAB for finite elements. And there's a lot of finite element code on the course page, ready to download. But now we have to understand it.

Are you ready to tackle, so in ten minutes we can get the central idea of finite elements in 2-D, and then after the quiz, when our minds are clear again, we'll do it properly Friday. This is asking you a lot, but can I do it? Can I go to finite elements in 2-D? OK. Oh, I have a choice. Big, big choice. I could use triangles, or quads. That picture, this picture would naturally set up for quads. Let me start with triangles. So I have some region with lots of triangles here. Got to have some interior points, or I'm not going to have any unknowns. Gosh, that's a big mesh with very little unknown. Let me put in another few here. Have I got enough? Oops, that's not a triangle. How's that? Is that now a triangulation? So that's an unstructured triangular mesh. Its quality is not too bad. The angles, some angles are small but not very small. And they're not near-- That angle's getting a little big too, it's getting toward a 180 degrees which you have to stay away from. If it was a 180, the triangle would squash in. So it's a bit squashed, that one. This one's a little long and narrow. But not bad. And you need a mesh generator to generate a mesh like this. And I had a thesis student just a few years ago who wrote a nice mesh generator in MATLAB. So we get a mesh. OK. Now, do you remember the finite element idea? Well, first you have to use the weak form. Let's save the weak form for first thing Friday. The weak form of Laplace. I need the weak form of Laplace's equation. I'll just tell you what it is. We all have double integrals. Oh, Poisson. I'm making it Poisson. I want a right-hand side. Double integral, this will be du/dx. d test function/dx. And now what's changed in 2-D, I'll also have a du/dy dv/dy, dxdy. That's what I'll get. And on the right-hand side I'll just have f, the load, times the test. dxdy. Yeah I guess, I have to write that down because that's our starting point.

Do you remember the situation, and of course that's on the exam this evening, is remembering about the weak form in 1-D. So in 2-D I expect to see x's and y's, my functions are functions of x and y. I have my, this is my solution. The v's are all possible test functions, and this hold for every-- This is like the virtual work. You could call it the equation of virtual work, if your were a little old-fashioned. Those v's are virtual displacements, they're test functions. That's what it looks like. I'll come back to that at the start of Friday. What's the finite element idea? Just as in 1-D, the finite element idea is choose some trial functions, right? Choose some trial functions. And our approximate guy is going to be some combination of these trial functions. Some coefficient U_1 times the first trial function plus so on, plus U_n times the n-th trial function. So this will be our-- And these will also be our test functions again. These will also be, the phis are also the V's. I'll get an equation. By using n tests, by using the weak form n times for these n functions. And so each equation comes-- V is one of the phis, U is this combination of phis. Plug it in. You get an equation. What do I got to do in three minutes, is speak about the most important point. The phis. Choosing the phis is what matters. That's what made finite elements win. The fact that I choose nice simple functions, which are polynomials, little easy polynomials on each element, on each triangle. And the ones I'm going to speak about today are the linear ones. They're like hat functions, right? But now we're in 2-D. They're going to be pyramids. So if I take, this is unknown number one. Unknown number one, it's got its hat function. Pyramid function, sorry. Pyramid function. So its pyramid function is a one there, right? The pyramid, top of the pyramid is over that point. And it goes down to zero so it's zero at all these points. Well, at all the other points. That's the one beauty of finite elements, is that then this U, this coefficient U_1 that we compute, will actually be our approximation at this point. Because all the others are zero at that point. So can you just, this is all you have to do in the last 60 seconds is visualize this function. Do you see it? Maybe tent would be better than pyramid? Either one. What's the base of the pyramid? The base of the pyramid, I'm thinking of the surface as a way to visualize this function. It's a function that's one here, it goes down to zero linearly. So these are linear triangular elements. What's the base of the pyramid? Just this, right? That's the base. Because over in these other triangles, nothing ever got off zero. Nothing got off the ground. It's zero, zero, zero at all three. If I know it at all three corners, I know it everywhere. For linear functions, it's just a piece of flat roof. Piece of flat roof. So I have, how many pieces have I got around this point? Oh, where is the point? I guess I've got one, two, three, four, five, six, is it? It's a six, it's a pyramid with six sloping faces sloping down to zero along the sides, right? So it's like a six-sided teepee, six-sided tent. And here are the, the rods that hold the tent up would be these six things. And then the six sides would be six flat pieces. If you would see what I've tried to draw there, that's phi_1. And you can imagine that when I've got that, I can take the x derivative and the y derivative. What will I know about the x derivative and the y derivative in a typical triangle? If my function is in a typical triangle, say in this triangle, my function looks like a+bx+cy. a+bx-- it's flat. It's linear. And these three numbers, a and b, are decided by the fact that the function should be one there, zero there, and zero there. Three facts about the function, three coefficients to match those facts. And what's the deal on the x derivative? It's just b. It's a constant. The y derivative is just c, a constant. So the integrals are easy and the the whole finite element system just goes smoothly. So then what the finite element system has to do, and we'll talk about it Friday, is gotta keep track of which triangles go to which nodes. How do you assemble, where do these pieces go in? But every finite-- every element matrix is going to be simple. OK.