MIT OpenCourseWare
http://ocw.mit.edu

18.085 Computational Science and Engineering I, Fall 2008
Transcript – Lecture 27

PROFESSOR STRANG: So, let's see, you probably guessed on that quiz problem three, it wasn't what I meant. I get a zero for that problem. But you'll get probably good numbers, so that's an election gift if it comes out that way. So they're all in the hands of the TAs to be graded. We have a holiday Monday, I think. We come back to Fourier. Now, so we just have a concentrated shot at Fourier, just about eight or nine lectures in November. So stay with it and that'll be of course the subject of the third quiz. Which will have no mistakes. It'll be solved by the TAs in advance and we'll spot things. So, and if we have the quizzes to return to you by Wednesday that will be great. I hope so, but they have a big job. A little bit, looking far ahead at the end of Fourier the quiz is December 4th, I think that's a Thursday. And that's the end of the course. So December 4th, so we'll be ending the course a little bit early. Because I'll be in Hong Kong, to tell the truth. And and we've done a lot, and with the review sessions we're really doing well. So, that's the future. Fourier, today is an important day too, finite elements in 2-D, that's a major part of computational science and engineering. The finite element idea, the idea of using polynomials, you can find in some early papers by Courant, a mathematician in New York, and by a guy in China neat guy named Fung Kong But those papers were sort of, you could do it this way if you wanted. It was really the structural engineers in Berkeley and elsewhere who made it happen ten years later. And the whole idea has just blossomed. Continues to grow. So I had an early book, in the `70s, actually, about the mathematical underpinnings. The math basis for the finite element method. And many other finite element books. Professor Bathe you know, teaches a full of course on that.

But, I think we can get the idea of finite elements here. We did them in 1-D, and now there's a MATLAB problem and I'd like to just describe that particular problem if I can, as an example. And, of course, you would use the code that's printed in the book, and that's available on the website just to download. But the problem is not on a square domain. It starts on a circle, so that the first lines of the code, the calling the MATLAB command square grid, are not applicable. So you have to create, then, a mesh. Well, I have a suggested mesh so I'll draw that, and then from that you want to make a list of all the node points. A list, P, of - so what the code needs is two lists. Well, let me draw a picture of, well, it's a circle. And I'm going to be solving Poisson's equation. The equation will be -u_xx-u_yy=4, in the circle. So it's Poisson but with a constant right hand side. That will mean that all the integrals of F times v, all the right hand side of our discrete equation will be, the integrals are all easy because we just have a constant there times the trial function. OK, and then on the boundary is going to be u=0. On the boundary. So it's a classic problem. And we can say what the solution is. So it's one with a known solution. I think it would be x squared. No, I guess one, one minus x squared minus y squared. This should all be on the .086 site.

I just didn't have a chance to look this morning to be sure it got up. So you can watch, here. So that, I hope, does solve the problem. Two x derivatives give us a two, two y derivatives another two, so we get four.

So we know the answer; the question is, and I'm interested in this question, for research reasons too, is what's the error when you go to a polygon? You go to a, these curved boundaries don't get correctly saved. You approximate them by straight lines. That would be the first idea. And with this, all this symmetry, let's keep the problem nice and use a regular polygon. So maybe I'll try to draw one with about eight sides, but, OK. So we impose u=0 at these nodes. So u is zero at those nodes, and then we have a mesh. So we want to create a mesh. OK, so with all the symmetry here, the natural idea would be to start with eight pieces, or M pieces if I have, this is a regular M side, let's say, and I'll take M to be eight in this picture. And I think we can work on just one triangle. By rotational symmetry, all those triangles are going to be the same. So I think our domain is really this one triangle here. That's where we're working. And in that triangle, I think we have zero boundary conditions. And across this edge I think we have natural boundary conditions. Slope zero, if I see the picture correctly. The rotational symmetry would mean that things are not changing. That every triangle is the same. So I think on these boundaries it's the Neumann condition, dU/dn=0. And I'm frankly not sure what to do at the origin, so I'll maybe just try both ways and see.

OK, so there is a real problem. Of course, it's artificial in the sense that we know the answer. But it's a real open question of what does the error look like, from doing that. So that's the goal and let me just say the problem I'll ask you to do, and it probably is quite enough to be ready for next Friday, is to use piecewise linear elements. Which is what I'm going to do. What every discussion of finite elements will begin with, linear elements. Those pyramids that I spoke about it at the end of last time. So that's what I hope, but actually I would be highly interested if anybody got into the problem, to try quadratic elements. So I'll just say here, second-degree quadratic polynomials would be more accurate. Would be more accurate. So, in other words, this is a first type of finite element called P_1, for polynomials of degree one. These guys, I would call P_2, for polynomials of degree two. And I've mentioned here the possibility of using quads. Instead of triangles, if I had squares for example, the simplest element would be a Q_1. So these are, if I manage today to tell you about how to use P_1 and P_2 and Q_1, you're on your way. And for the requirements of this course, P_1 is the first point to understand. OK. While I'm speaking about codes and meshes, let me draw the mesh I proposed in the homework problem. So I thought OK, we just have to have a simple mesh here. So let me draw that line in. I know all these points, right? This is 0, here. And that point's on the circle. And so is this, that point might be, so what's the angle there? That angle is probably pi/8. The whole angle would be 2pi/8, and eight of them would go all the way around. So I think that angle is pi/8.

And so this point would be cos(pi/8). sin(pi/8). We know where they are. But that's going to be what we have to list. We have to list where are the coordinates of all the mesh points. So let me describe the rest of the mesh points and then see what that list would look like. And once you've created the list, the code will take over. And then you plot the results and see what's going on. So let me suggest a mesh. It's pretty straightforward. I just divided this piece into N, this center thing into N, so I called that distance h. So Nh gets me out to here. Whatever that is, that's cos(pi/8), I guess, that's the x coordinate of that line. So those are mesh points and then let me keep drawing these. So these will be mesh points too, and these will be mesh

points too. So at this point, I've got probably 12 or 13 mesh points. But I've got quads, right? Well, I've got a couple of triangles here that I'm not going to touch, those are fine. But these are quads and they could be used with the Q_1 element, but I'm thinking let's stay with triangles. So I just suggested to put in triangles, put in these diagonals, keeping symmetry. And so there's the mesh. There's the mesh.

And then what does the code ask for? So it's got 13 mesh points. And the code, first of all it wants a list of the coordinates of all those mesh points. So that the code will, and I better number them, of course. So let me number them one, shall I number them the center guys first, one, two, three, four, five, and then up here six, seven, eight, nine,, now I don't know if this is a good numbering. Ten, 11, 12, 13. Why don't we, just to have some consistency within plans, why don't you, don't have to take N=4. I hope you'll take, what did I take N as four or five? Yeah, right. N is 4. But you'll want to try different N's. N=4 would be a good crude start to see what's going on, but then I hope you'll go higher and get better accuracy. And you can see how the accuracy improves, how you get closer to that as n gets bigger. OK, but got the nodes numbered. Oh, I better number the triangles. OK, how shall we number the triangles? Shall we do along the top, or this? I don't know. What do you want to do for the numbering of the triangles? Maybe run along the top, and then run along the bottom, because then it'll practically be a copy. So I didn't leave myself much space, but one, two, three, four, five, six, seven. Seven triangles along the top and seven along the bottom, so I have 14 triangles in this mesh. So it's a mesh with 14 triangles and 13 nodes. And I know the positions of everyone, right? I know the x,y coordinates of every one. So what the code will want is a list of those coordinates. So a list, P, P will be a list of coordinates. The first guy will be of, the nodes so 13 rows, three columns. So it's a little 13 by three matrix that tells you where all the nodes are.

So the first one on that list would be (0,0). that's for node one. And the second one would be whatever the coordinates of that are, something zero. (h,0), I guess it is. The third one will be (2h,0), and so on, and then complete the list of 13 positions. So you are then told the code where all the nodes are. What else do you have to tell it? Not much. You now have to tell it about the triangles. So now for every, why do I say three columns? Maybe only two, is it? You see the point already. I've forgotten, maybe, yeah. I don't see why. Well for triangles, I'm going to need three, for nodes maybe it's only got two. Maybe it's 13 by two. I don't see why I need three. But, anyway. Then the other list is triangles. So this will be the list, t, and so it takes triangle number one. Which is right here. That very first triangle. And what does it have to tell us about triangle one? The three nodes. If it tells us the three node numbers, and this list, P, gave their positions, we've got it. So how many triangles did that I have? 14? So t will be 14 by three, and so the first guy will be just one, node number one, node number two, and node number six. That will tell us which is the first triangle. And the second triangle, I guess I've drawn from two. Two, seven to six, right? That's a very skinny triangle up there but it's the one that started at node two, went up to seven and back to six.

So a list like that. And then the code will do the rest. I hope. Almost all the rest. The code will create the matrix K. It'll create the matrix K with, it'll be singular. Boundary conditions won't yet be in there. And then a final step after that K, or maybe we could call it K_0 is created, a final step will be to fix u. At least at these three points. So these three will be boundary nodes. And as I say, I'm not too sure about that one, I apologize. At those boundary nodes I'm going to take the values to be zero. So this is going to be zero along the whole edge, because if it's zero there, zero

there, zero there and zero there, and if it's linear, it's zero. So the final, sort of, subroutine in the code, the final group of commands you want to impose, zeroes here, that should then make the matrix K invertible, and then you've got KU=F to solve. So what the code is doing is creating K and F. You see the overall picture? I jumped right into this particular mesh, particular problem, but now I really should back up to where it starts. This this is going to be the weak form of Laplace, Poisson, maybe I'll make a little space to put in Poisson's name too. You have a picture already of what this weak form is about, so now I'm really backing up to the start. I take the equation and I get its weak form. And remember that's in the continuous case, as it was in the quiz problem. The first step is the continuous weak form, and then the second step is choose test functions and, trial - I'm sorry, gosh, I'm in bad shape here. Because they're the same. This isn't my worst error, today. But those are trial functions, and these are test functions. OK, questions at this point, because I've, yeah, thank you. Good. Let's look at this picture.

AUDIENCE: [INAUDIBLE]

PROFESSOR STRANG: Because, we did. That's right. So because my question is what is the continuous problem, I would like to solve has u=0 on the polygon. So in a way you can forget the circle, where we know the answer now. We really are looking on the polygon, and I would like to know what's the solution like on that polygon. And then so there are two steps, the first step was start with a circle, we have the answer. Second step is go to a polygon, continuous problem, Poisson's equation in the polygon. How different is that from this? Because this will not satisfy the polygon boundary conditions. So that's the circle answer. Then the question is, what's the polygon answer. And I don't know that. You may say a regular polygon, you can't do that. I didn't think you can. It's amazing, but probably a triangle or a square. So if M is three or four, probably some formulas would be available. But I think once we get higher, I don't know the answer to the Dirichlet problem, to Poisson's equation on a polygon. On a regular polygon and that's what I would really like to know more about. And how do I find out more about it? By finite elements. With your help. Taking that polygon, breaking it into a mesh, looking only at one triangle just for simplicity, and getting u finite elements. Well, I should say u_p_1. That's the finite element solution using linear. I would really like to know u_p_2, the finite element solution which will be better. If I use quadratics. So now I get the fun of describing the linear elements, the quadratic elements, the quads.

But did I answer that question OK? Yeah. So this is the problem I would like to know the answer to. If I have this equation, zero boundary conditions on a regular polygon with M sides, what's the answer? And it's going to be close to this, but it won't be the same. Because this does not vanish on the polygon edges. And I would like to compare the slopes, too. So the homework problem asked you not only to compare u circle with u_p_1, but also the slopes. The slopes here are easy, slopes here are easy because it's a bunch of flat functions. So the slopes are just constant in each triangle. OK, I'm guessing that the error gets smaller as you go in. I think that if you plot the error, it'll be largest out here and get small there. But remains to be seen. So I hope you enjoy - yeah, good.

AUDIENCE: [INAUDIBLE]

PROFESSOR STRANG: Rather than seven.

AUDIENCE: [INAUDIBLE]

PROFESSOR STRANG: No, the middle. There's nothing magic about any particular mesh. I just chose this mesh as pretty good, and actually, I'm imagining M could get pretty big. That would be interesting. M=8 would be interesting, M=16, M=1,024, now then I'd really get interested. OK, but so if M is 1,024, then this side would be very small. Right? And I just wanted more triangles. Actually, I would like more than I've got. I'd like, if M was really big, then probably N should be at least that big. So I should have a thousand this way, if this is just a tiny bit. I just want little tiny h, and then, yeah. Actually, that might not be too bad. If m anM N were roughly comparable, then that length would be roughly comparable to these lengths. And the triangles would be pretty good shape. And that's what you're looking for. I think there's a lot of experiments to be done here. So, I'm thinking then of M and N. Here I took N to be just four. When M was eight. That's fine. But if you keep M and N roughly the same size, then you've got triangles that are not too long and skinny. I'll tell you when you might want. So generally you want nice shaped triangles. You don't want angles very small or very large, usually. But there would be, anybody in Course 16 can imagine that if I am computing the flow field past a wing, that long, thin triangles in the direction of the wing are natural. I mean, somehow a problem like true aerodynamics is by no means isotropic. I mean, the direction of the wing is kind of critical to whether the plane flies, right? So don't make the wing vertical. And if you want accuracy, then you have long, thin triangles in the direction of the flow.

But here we're not doing a flow problem. We haven't got shocks, or trailing edges, and other horrible stuff that makes planes fly. We just got Poisson's equation. OK. Thanks for those good questions, another one.

AUDIENCE: [INAUDIBLE]

PROFESSOR STRANG: What would the dimension look like? Ah, would you like me to show you something about quadratics? Yeah. Shall I jump into quadratics, it's kind of fun. Quadratics, so let me just do, so I'll come back to the weak form, right it's totally, oh I'll do it now. It's so simple I don't want to forget it. The weak form, so I write the equation down, -u_xx-u_yy=f. This is the continuous weak form equal f(x,y), OK? So that's the strong form. And I've made it the Laplace in here to keep it simple, on any right hand side. OK, how do I get to the weak form? Just remind me, I multiply both sides by any test function v(x,y). Multiply by v(x,y), and then what do I do? I integrate over the whole region. So that's the weak form, dxdy, this is for all v, all v(x,y), all, I'll say all admissible v(x,y). So that's the weak form. If this holds for all this great family of v's, the idea behind it is, that if this holds for all these trial functions, test functions, v(x,y), the only way that can happen is for this to actually equal that. That's a fundamental lemma in this part of math, and of course it has to be spelled out more than I'm doing in words. But the idea is that if these hold for such a large class of v(x,y), then the only way that can happen is for the strong form to hold. For this to actually match this. OK, so that's the start. But then what's the next step in the weak form? I I like the right hand side but I'm not so crazy about the left hand side. I'm not crazy about it because this says second derivatives of u, and my little roof functions, pyramid functions, haven't got second derivatives. So I would be dead in the water without doing the natural step that makes everything beautiful, which is? Integration by parts.

Integrate by parts. Move derivatives of of u, on to v. One derivative onto v, off of u, so then u and v each have one derivative. I can use my piecewise linear, piecewise quadratics, all my finite elements are going to go fine. So I integrate by parts. So

integrate by parts, and what is that mean in 2-D? Of course I have a double integral here. So integrate by parts, that mean you the Green's formula. That was the key point of this Green, or Gauss-Green's, formula. Can I do it first in, this is -div(grad u), times vdxdy, we can write out all the terms. We can use vector notation. I could use that nabla, that upside down triangle notation, or whatever. But maybe good to see it a few different ways. So what's the point? When I integrate by parts, that minus disappears to a plus, I have a double integral then, and these derivatives move off of, I'm taking one derivative off of here, the divergence moves over there, but when the divergence moves onto v it becomes? The transpose. It becomes gradient. And so this is gradient view, gradient of v. dxdy, plus boundary terms. The integral of, what is, let's see. What do I have in this integral, I have grad u dot n, times v around the boundary. And that's with my boundary conditions that's going to be gone, so I can come back to that.

Now, you all looked a little uncertain when I wrote Green's formula this way. For this problem I can write it more easily. This is my left side. I want to write the answer, I just want to write this weak form in a much simpler form. So let say, what have I got here. Well, all I've got is one derivative is moving off of u and on to v. And the minus sign is disappearing, so I have du/dx times dv/dx. Right? One off of u, onto v. The other term, one y derivative, moving off of this and onto v. Minus sign again going to a plus. du/dy, dv/dy. That's the integral. That's it, that's cool. Easy to do. And on the right hand side of course I have no change. The integral of f(x,y)*v(x,y)*dy. Now, that's the weak form, dx/dy. Here it is, weak form. That's pretty nice. Beautifully symmetric, though the matrix that comes up when we plug in finite specific trial functions and test functions is going to be a symmetric matrix K. And the integrals of first derivatives, so as long as our functions, our trial functions and test functions are continuous, that is, they shouldn't jump. If the trial functions or test functions jump, then if I have a jump, then the derivative would be a delta. I'd have another delta here, I'd have an integral delta, a delta times delta, and I don't want that. That's infinite. Those discontinuous elements would not be conforming, and that's a whole new world of discontinuous Galerkin. I'd have to impose penalty stuff, and Professor Peraire I mentioned. And others, Professor Darmofal in aero are experts on this. We're doing continuous form. CG. Our piecewise linear, piecewise quadratic, they'll be continuous. All I have to do is these derivatives. Integrate those things and that's what the code will do.

OK, I've got to the weak form. That's the weak form. Now comes the finite element idea. So there is our weak form, now ready for the finite element idea. OK, so what was that idea? That's the continuous problem. Now, the finite element idea is, plug in U as a combination. Let me write out the terms. You know what's coming here. If I'm using finite elements, I'm going to choose nice polynomials, phi, say, N of them. That would be like, one for every node, so I would have 13 functions here. I'm going to choose the v's to be the same as the phis. And then, I'm working then in 13 dimensions instead of infinite dimensions. So what do I do? For this limited subspace, this finite element subspace, this piecewise polynomial, piecewise linear subspace, I plug that into the weak form and I test it against 13 V's, which are phis. So I plug that in, so now what is K? Now let me just say, so I now have the integral of, yeah I guess I'd better plug it in. K_ij would then be the integral. I'm just copying the weak form in. Of dU/dx, no, sorry I'd better just plug it in first. dU/dx*dV/dx plus dU/dy*dV/dy, those are the integrals I have to do. And on the right hand side I have to do the fV. OK, plug that in. That's the integral over the whole domain. When I plug it in this U is a combination of known functions and the V's will be the same

guys. So what am I going to get here? It's just as in 1-D. So no new ideas entering here. The new idea's going to enter when I construct these phis.

Let me just say, though, one thing. In 1-D, we've pretty much had a choice of, when it was one dimension. Just remember that. In one dimension, when I had these hat functions, when I had these guys, integrated against these guys, I pretty much had a choice of did I want to think about integrating that hat function against that one. Or actually it was their derivatives. It was the integral of U, yeah. Of phi, what I needed was all the integrals of phi_i', phi_j'. Those are what I needed, these go into K. Into the matrix K. In fact, that's what equals K_ij, the integral of phi prime. In 1-D. OK. Now, what I was going to say, I could do it this way if I wanted. But you remember the other way to do it? Was elements at a time. So this was one method here. That found the entries of K separately, one by one. The other way was take the elements, one by one. So the other way was take an element like this element. It's got two functions, two trial functions are involved there. There's a little two by two, so this is four. Two by two element matrices. K equals. And the quiz recalled that part. That approach. So what I want to say is that's the right way to do it in two dimensions. A triangle at a time. That's the way the code will do it. It creates these little element matrices, and then it stamps them into the big matrix K. Alright. So I want to do this integral one triangle at a time. Is the good way. OK, and that's what the code will do. Actually, I think that the best way to learn these steps is just to read the lines of the code. You can read them in the book, Page 303 or something. And you'll see it just doing all the steps that need to be done. One triangle at a time.

So, now. Now comes the fun. I get to answer what do these piecewise linear elements look like. What do the quadratic elements look like. What do the Q_1 quad elements look like? This was the golden age of finite elements, when people invented these ways to create piecewise polynomials. And it continues. People are still inventing, I had a email this week, somebody says I've got spectral elements. People are going higher and higher degrees. You, know sixth degree, eighth degree. In order to get more accuracy. OK, let's start with P_1. How do I describe a P_1 element inside a triangle? So in a triangle, the unknowns will be the value, this has a height U_1, this has a height U_2, and a height U_3 at those nodes. Inside the triangle, the function U is linear. a+bx+cy. Then, you see that if I know these three values, then I know these three numbers. And vice versa. There's a three by three matrix, right? There has to be a three by - any time you see pictures like this, this is like the good part of 18.085 is to realize that if I have three numbers here, three values and I've got three coefficients, that there's some three by three matrix that connects them that you're going to need. That's like a meta-message of this course. Is, you've got to translate between the node values and the coefficients. Because the node values are the unknowns, right? These are the guys that are multiplying the pyramid function, this is multiplying a pyramid function with height one. At that point, going down to zero, so this one will be a pyramid function of height U_2 times one, going down to zero. And U_3. So we've got a flat function in here. And it looks exactly like that. OK?

So what do I want to say? When we know the positions of these three nodes from our list, P, right? These were the crucial things we did. The positions of all the nodes, we know where they are. Then there has to be a three by three matrix that will now connect to the coefficients. Why do we want the coefficients? Because those are what we do when we integrate. The coefficients are what we need, we need to integrate dU/dx, dU/dy, dU/dz. Sorry, dU/dx, dU/dy. Are you visualizing this overall solution capital U, yeah. So what the overall solution capital U, you should visualize with a

combination of all the little u's, is zero here and then it's going to go up and these triangles and bend around and, I don't know, maybe down again. Or maybe, no, maybe it keeps going up. This is probably the largest value, because it's the largest value and the correct solution. So is this is probably going to be the highest point of this. What's the Forbidden City, right? In China, is in Beijing is like, or a single, do pagodas have flat? No. We we will meet pagoda functions. But this would be just an ordinary western roof, I guess. Just flat pieces. Yeah. OK, see, you've got to see the whole thing and then you look at each piece. Each piece looks like that, and the integrals are doable.

OK, so while I'm going here, I want to do quadratics. You'll get the idea right away. So, same triangle, now I'm going to have quadratics. So I'm now going to have, so this won't be the arrow, this arrow will now go this way. I'm going to have dx squared, exy, and f y squared. So now how many coefficients have I got to determine a quadratic? Six, right? a, b, c, d, e, f. How many nodes do I need? Six. Where are they? Well, the natural positions are those guys in the mid-point. So now, those are all nodes now. Some nodes are at vertices of triangles, some nodes are at midpoint. But remember, we've got other triangles hooking on here, many other triangles, all with their own six nodes. Well, not their own, because they share. That's a big point. So there's a grid of triangles, with nodes for quadratic. And we've got one, two, three, four, five, six, seven, eight, nine, ten, 11, 12, 13, 14, 15, 16 nodes, I think. And within each triangle, this is what we've got. So there's a six by six matrix for each triangle. A six by six matrix which will connect the values U_1, U_2, U_3, U_4, U_5, U_6 for this triangle. Connect those six heights with these six numbers. And what will the roof look like within that triangle? Well, sort of curved. A parabola, right? A parabola somehow in 2-D, it'll look like this, yeah. Yeah. And here's the key question. Will that roof, that curvy roof, fit the one over there? Because if it didn't fit, we're in trouble. This derivative would have a delta function, and we've got delta functions, and integral squaring them would give infinite.

So here's the question. Why does this roof, using these six points, fit on to the roof that uses U_7, U_8, U_9, and U_3 U_4 and U_6? Why do those two roofs fit together? This one piecewise polynomials? Of course, the slope will change. But the roof won't have a gap. Water won't go through it. Why's that? Do you see why? Because what do they share, what do those two curvy roofs share? They share a side. They share the same values along the side. And are those three values that are shared along the side sufficient to make it match all along the side? Yes. That's the important question. Finite elements lives or dies on that question. The answer is yes, because along that side, if I just focus on that side, where these three values are shared on both sides, by the triangle on both sides. Along that edge, what kind of a function have I got? It's second degree. This is whatever, when I restrict this to just run along a line, it's a parabola. And the parabola is determined by those three values. So having it right at three points means I have it right the whole way. Yeah. So there you see what quadratic elements would look like, and you could extend the code in the book and on the CSE site to work for quadratic elements. And you want to just guess what cubic elements could look like? I'm sorry, we've run five minutes over, but maybe finite elements is worth it. So if I had cubic elements, any idea how many? So I'm now going up to, I'm adding g x cubed, h, i, j, any idea how many coefficients I now have? Four new ones plus these six is ten. I need ten nodes. Where I am I going to put ten nodes in this triangle? I want to put them, I'd like to have some on the edges. Because the edges help me make triangles match each other. They'll just be like bowling balls. So here's six, oops, that wouldn't be believable. Is that right? Four, three, two, and one. Yeah. Yeah. OK.

So, now I've got a bubble node inside and I've got four nodes of vertices and two points, at two 1/3 points, and that will then match the triangle next to it. Because four points determine a cubic. There you go, I hope you have fun, I hope you have a great holiday. I'll see you Wednesday for Fourier and always open for questions on the MATLAB.