PROFESSOR: I thought I would, in this last lecture before the break, speak about one specific topic. It's often referred to as a Fast Poisson Solver, so what does Poisson mean? Poisson means Laplace's equation. So, this is the five-point Laplacian, which could be some other discrete Laplace matrix, but let's take the one we know. So we're in two dimensions, and you use the word Poission's name, instead of Laplace's name, when there's a non-zero right-hand side. So otherwise, it's a Laplace Solver, but here Poisson. OK. So, remember that right-hand side comes from maybe a right-hand side f of xy in the differential equations, but it also comes from non-zero boundary conditions, because non-zero boundary conditions, when the five-points hit a boundary, that known value was moved over to the right-hand side and becomes part of f. So f comes from the non-zero boundary conditions, as well as the non-zero right-hand side. OK.

Important problems, but special on a square or a rectangle or a cube or a box, so we're speaking about special geometry. Today as we've been doing, I'll take the case on a square, and you will see that the whole idea would not fly, if we were on an ellipse or something like that. But a lot of problems on rectangular domains do appear in applications, or we could use, I hope you always think now about possible preconditioners. Any time you have something fast, it's a candidate to be a preconditioner for a real problem. The real problem might not be on a square, or the real problem might not have the constant co-efficient that we have in the Laplacian. In that case, you're not solving the exact problem, but one that could be reasonably close to it. OK. So we've discussed the solution to this problem, and actually I have a little more to say about the movie that is now on the website, because I'm quite excited about that movie. I'll come to the movie in a second.

Let me just say what today's lecture would be. The key idea here is that the eigenvalues and eigenvectors of this giant matrix of order n squared by n squared,

sizes n squared by n squared. The eigenvalues and the eigenvectors are known. First of all, they're known. The eigenvectors are nice discrete sine functions; they're sines, because I'm assuming they come to zero at the boundary; so that's why I have a sine, rather than a cosine. First they're known, and second we can work with them very quickly, using the FFT. So the point is that it's quite exceptional; in fact, I don't think I know any comparable example, in which a linear system is solved by using the eigenvalues and eigenvectors. Usually eigenvalues and eigenvectors, they pay off for differential equations that are growing in time. Then it is worth computing them, because then you can just multiply by either the lamda t, and you know what's happening later. Eigenvectors, eigenvalues have other purposes, but very, very rarely are they used to solve a linear system. I mean, it's usually far more work. And of course, it would be incredibly more work if we had to find the eigenvalues and eigenvectors, but for this problem we know them.

And it also would be incredibly more work if the matrix of eigenvectors, the basis matrix, the key matrix that I'll denote by s -- so the eigenvectors go in a matrix s; the eigenvalues go in a matrix capital lambda; so we know lambda. It's going to be a simple matrix, just diagonal, got those n numbers and n squared numbers, I guess, because we're of size n squared. But these eigenvectors we know. So we know them, and we can compute quickly with them, using the FFT, or using, you might want me to say, fast sine transform, discrete sine transform, rather than Fourier transform, which we think of as doing the complex exponential. So this is a small special fast Fourier world. It's a special fast Fourier world, in which the FFT and the related sine and cosine give us a quick answer, faster than elimination, because you all know, it's n log n, if n is the number of Fourier components, and that's hard to beat.

Then I'll mention also, for this same problem, there is another way in which it can be simplified. It's not a Fourier way, just a direct combining neighboring equations, so that'll be number two. OK, but mostly the lecture is about number one. The best example I know in which you would use eigenvectors/eigenvalues to solve an ordinary linear system, and I'll say in a word just how you do it, and then what these

eigenvectors are. OK.

Pause. Time out to say something about the movie. So that movie that's now on the website does sparse elimination, and the example it takes is k2d, and you can make it 8 squared by 8 squared, 10 squared by 10 squared, 20 squared by 20 squared, because it shows the order that the nodes are eliminated and the graphs of non-zero's in the matrix. It's a bit slow. If you do 20 by 20, you have to go away for lunch, well maybe not lunch, but at least coffee before it's done, but when it's done, it counts the number of -- it shows the non-zero's that are in the elimination in the factor l from elimination, and it counts the number of non-zero's, and I was just talking to Persson about also getting it to count the number of actual [? flops ?].

Well, why am I interested? I'm interested, because I don't know what power of n those numbers are going there. I don't know whether the number of non-zero's, and I did 10, 20, 30, and it looked not too far from the power capital n cubed, but the notes say for nested dissection, which would be another ordering n squared log n. So, and of course, what power we get depends on what algorithm we use for ordering, so nested dissection is one ordering, which hopefully we could put into another movie. The movie now has exact minimum degree, real MMD, which takes as the next node the one with absolutely the minimum degree, not just close to it. Anyway have a look at that movie, and if you have any interest, see how the count increases as n increases, and also you could change, slightly adapt, the algorithm that creates the ordering, creates the permutation and see what happens there. There's a lot to do with that, and it's a pretty fundamental problem. Actually, we're talking there about what MATLAB's backslash operation will do for this equation. So MATLAB'S backslash operation will use elimination; it won't use fast Poisson solvers, but now let me come to the fast Poisson solver. OK.

So I guess the main point is I have to say what are the eigenvalues and eigenvectors, and how do they get used. So let me say, how do they get used. So how can I use eigenvalues and eigenvectors to solve a problem like that. Let me just call it k rather than k2d, sorry. ku equals f, OK, by eigenvectors. OK, there are three steps. Step one. Expand the right-hand side as a combination of the

eigenvectors. OK. So expand f, this right-hand side vector, as some combination of the first eigenvector, maybe I'm going to call it y1. Second eigenvector, nth eigenvector, OK, good. That means what do I mean, I mean you have to find those c's, that's the job there. Find the co-efficients. So that's a job, a numerical -- it's a linear system to solve and we'll see what it amounts to. OK, but suppose the right-hand side is a combination of the eigenvectors, how can you use that? Well, step two is the real quick step. Divide each ci by the eigenvector, eigenvalue, lambda. OK, so it's my eigenvector, so I'm assuming that kyi is lambda iyi and that we know these guys. So this is known. And now the question, I'm just saying, how do we assume known?

So my question now is how do we use it? OK, step one, the idea is going to be write everything in terms of eigenvectors. Work with the eigenvectors, because if I've got eigenvectors, the step is scaler; I just divide these numbers by those numbers, and then I've got the answer. And then construct, the correct answer will be u, will be c1 over lambda 1y1 and c2 over lambda 2y2 up to cn over lambda nyn, a combination of those same eigenvectors with the same coefficients, just divided by lambda. But this is, of course, another numerical job; this is like adding up a Fourier series; this is like finding the Fourier coefficients, this is like reconstructing the input, only because I've divided by lamba i, I'm getting the output here is u, when the input was x. And do you see that that is the correct answer? All I have to do is check that ku equals f, so check that this answer from step three is the right answer. OK, so I multiplied by k. When I multiply y1 by k, a factor lambda 1 appears, the eigenvalue; it cancels that; well that's y divided, so it would cancel, and I have c1y1. When I multiply this by k, ky2 is lambda 2y2; cancel the lambda 2s, and you're left with c2y2, and so on. So, when I multiplied by k, I got f. That's it.

So that's the whole idea, written out, but now what actual computations go into steps one and three? Step two is pretty simple. Well, actually this is a good way to look it. I want to write that same algorithm in matrix language. OK, so in matrix form we have the matrix of eigenvectors, and that's what I'm calling s. And it's got the eigenvectors y1 y2, yn, and it's columns, and the eigenvalue matrix, we need a name for that too, and that we decided to call lambda, and that's got the

eigenvalues on its diagonal. So this is 18.06 linear algebra. The matrix of eigenvectors if I multiply k by s, then I'm multiplying k by all its little eigenvectors, and k times this gives me lambda 1y1, k times y2 gives me lambda 2y2, kyn is lambda nyn, and if I look to see what this is, this is the same as y1 to yn multiplied by lambda. If I just multiply [UNINTELLIGIBLE] on the right by lambda, it will take lambda 1 times the first column, lamda 2 times the second lambda n times the third, which is, times the last, which is what we want, so it's s lambda. This is all n eigenvalues and eigenvectors in one matrix equation, that's all that is. It's just ks equal s lambda i just says, this for all i at once, all i at the same time. OK.

So if I use these matrices in describing steps one, two, three, I'll see what's happening matrix language. OK, let me just do that. Step one, step one is looking for f as a combination of the columns of s. So step one is just f equals s times c. The vector of coefficients multiplies the columns of s and adds to give f. Then step two, which just divides everything by -- divides by the lambdas. Step two just creates lambda inverse, sc. So I took what I had -- let's see, no, the lambda inverse better be multiplying the c. Well, actually I can do it all in -- well step two, that's the easiest step, I should be able to do it. c is changed to lambda inverse c, so that c becomes lambda inverse c, OK. And then step three uses lambda inverse c to construct u. So step three is the answer u, what do I have here? I've got a combination of these vectors, so they're the columns of s, and what are they multiplied by? They're multiplied by the c's over lambdas, which is what I have here. That's s lambda inverse c. Those are the three steps.

And what's the work involved? Here, the work is solving a linear system with the matrix s. Here, the work is taking a combination of the columns of s, doing a matrix multiplication. Those two steps are usually full-scale matrix operation, and of course, the s , if I just complete this, I'll see that I get the right thing that's s lambda inverse and c is s inverse s. There's the answer. u is s lambda inverse, that's a lambda inverse there: s inverse f. That's the correct answer in matrix language. Right. This is k inverse, that's k inverse. k is s lamda s inverse, and if I take the inverse of that, I get s lambda inverse, s inverse to multiply f. Well, I doubt if you're

much impressed by this lower board, because the upper board was the same thing written out. It took some indication of what the separate pieces were, but it's pretty clear.

OK, now the million dollar question is, is it fast? And the answer is, almost certainly no. But for the particular matrix s, which by good fortune s could also stand for sine, this matrix of eigenvectors for this particular problem are sines. These are the discrete sines, so this is the discrete sine transform. That's we're doing, we're doing the discrete sign transform, because those discrete sine vectors are the eigenvectors of k. OK, now let me say what that means. First I'm thinking of k in 1d. So this is my 2 2's and minus 1's and minus 1. Its eigenvectors are discrete sines, if I multiply that by sine kh -- well, let me just take the first one, sine h, sine 2h, sine n minus 1h, that will turn out to be an eigenvector. So this is ky, ky1, the first eigenvector. The eigenvectors are -- let me draw them. The eigenvectors for that second difference matrix k are -- here's the interval, 0 to 1, I chop it up in steps of h, and I plot the sine, which starts at zero and ends at zero, because those are the boundary conditions, and here is sine h, sine 2h, sine 3h, sine 4h, sine 5h, so for the five by five case -- maybe I should just be using n here, or maybe I should even be using capital n, so capital n is 5 in this example. Good.

What's on the other side of that equal sign? Some eigenvalue times the same vector, sine h, sine 2h, down to sine nh, and that eigenvalue -- oh, let me just write lambda 1 for it. We know what it is. The fact that this is an eigenvector is just trig; you know, I multiply minus 1 of that plus 2 of that minus 1 of sine 3h, and I use a little trig identity. So minus that, 2 of that, minus that, turns out that to be a multiple of sign 2h. Well sine 2h give us a 2, and then the minus sign h and the minus sine 3h combine into sign 2h times, I think, it's s cosine of h or something, it's that eigenvector that's near zero. But the cosine of h is near 1. Does that look familiar? That combination of sine h and sine 3h should give us twice sine 2h times some cosine, yep, Elementary trig identity. OK, so those are eigenvectors. That's the first one, the next one would have h times the k'th one would have h times k instead of just h itself, it would take jumps of every k sine, and then a cosine hk would show up there, and this would still be an eigenvector. OK. I'm making a little bit explicit these

vectors, but the main point is they're sines, they're discrete sines at equally spaced points.

That's what the real version of the FFT just lives on, and it would also live on discrete cosines if we had different boundary conditions, we could do those, too. So this isn't the only -- these zero boundary conditions are associated with the name of Dirichlet, where zero slopes are associated with the name of Neumann, and both, this one gives sines, Neumann gives cosines, the FFT deals with both. OK. So, that's the fast solution, and it would take n squared -- well I have to go to 2d. sorry, I guess I have a little more to say because I have to get from this one-dimensional second difference to the 5.2-dimensional second difference, and that's what's going to happen over here. I wrote up some stuff about the Kronecker operation, which is the nifty way for these special problems to go from 1d to 2d. You remember the deal, that k2d, our 2d matrix was this Kronecker product of k and i, that gave us second differences in one direction, and then we have to add in the Kronecker product of i and k to get second differences in the other direction. And then we better print, because that matrix is going to be large, I don't want to [UNINTELLIGIBLE] it, yeah. What's the point?

The point is that if I know the eigenvectors of k, then I can find the -- if I know the 1d eigenvectors, I can find the 2d eigenvectors, and you don't have to know Kronecker products to do that. All you have to do is just make a sensible guess, so the eigenvectors in 2d, have a double index, k and l, and their components are sines in one direction times sines in the other direction. So what are those sines, there's a kh, I guess, lh. Those are the first components, I guess I have to tell you what all the components are: k, the seventh component in the x'th direction, there'd be a factor 7, so kmh sine lnh. This is the mn component of ykl. It's just what we had in 1d, in 1d there was no l, the components were just sine kmh. Now we've got two, one in the x direction. These are the analogs of the -- the continuous case would be sine, k pi x, times sine l pi y. Those are the eigenvectors as eigenfunctions, functions of x and y. And the point is that, once again, with these beautiful matrices, I can sample these at the equally spaced points, and I get discrete sines that the FFT is ready to

go with, OK. I'm giving this much detail partly because the continuous case, of course, our operators d second by the dx squared and d second by dy squared, and the whole idea of separating variables, of looking for solutions, u -- here is the eigenvalue problem, the continuous eigenvalue problem. The Laplacian of u, maybe I do a minus, the Laplacian of u equal lambda u, and that's a partial differential equation, usually it's not easy to solve, but if I'm on a square, and I have zero boundary conditions, then I've solved it, by separation of variables, a function of x times a function of y. And that function of x times function of y is exactly what Kronecker product is doing for matrices, yep. I thought maybe this is good to know when the problem is easy, and as I say, the possibility of using the easy case on a square for preconditioning a not so easy case is attractive. All right, so that's what I wanted to say about number one, that's the main suggestion, and again, the point was just to take these three simple steps, provided we know and like the eigenvector. Here we know them and we like them very much, because they're those discrete sines.

OK, now just to finish comes, what's up with odd even reduction. I'll use the same 1d problem first. It works great in -- that's not good english, but it works very well in 1d odd even reduction, you'll see it, you'll see, oh boy, simple idea. But of course, don't forget that ordinary elimination is a breeze with a tri-diagonal matrix, so nothing I could do could be faster than that. But let's see what you can do. I just want to write down the -- OK, keep your eye on this matrix, so I'm going to write out the equations. So it'll be minus u, i minus 2 plus 2ui minus 1 minus ui, that would be fi minus 1; that would be equation number -- i minus 1, right with a minus one, two minus 1, centered there; and then the next one will be a minus ui minus 1 plus 2ui -- I better move this guy over further -- minus ui plus 1, that will be fi, right? That's equation number i, and now I just want to look at equation number, the next equation, minus ui plus 2ui plus 1 minus ui plus 2 is fi plus 1.

So I've written down three consecutive equations, three consecutive rows from my simple matrix: a row, the next row, and the next row. So the right-hand sides are coming in order, and the diagonals are there, and if I look at that, do I get any idea? Well, there is an idea here. I'd like to remove these guys, the ui minus 1's and the ui

plus 1's, so that's where this word odd even reduction is coming in. I'm going to reduce this system by keeping only every second unknown and eliminating those. How to do that? Well, you can see how to eliminate, if I'm multiply this equation by 2. If I multiply that middle equation by 2, that becomes a 4, this becomes a minus 2, this becomes a 2, and now what shall I do? Add. If I add the equations together, I get minus u, i minus 2, these cancel, that was the point plus 4 minus a couple, so that's 2 ui's minus this guy, ui minus 2 is that sum fi minus 1, 2 fi's and fi plus 1. Well, sorry it's squeezed down here, but this is the point.

The main point is look at this. What do we have? We've got a typical equation, but now we've removed half the unknown. The problem is now half-sized. We've only got the even-numbered unknowns at a small cost in updating the right-hand side, and the problem's cut in half. So that's this odd even reduction, and it cuts a problem in half, and everybody knows right away what to do next. The one mantra of computer science, "Do it again." So that is the same problem on the even, we do it again, so I should really call it cyclic reduction, we just cycle with this reduction, and in the end, we have a 2 by 2 problem. That seems pretty smart, pretty successful move, and I guess if we do an operation count, well, I haven't thought that through. What does the operation count look like? It must be pretty quick, right? Well, undoubtedly we're solving this linear system in o of n steps. Well, no big surprise to be able to deal with that matrix in o of n steps, because elimination would take o of n steps, it's size n, but it's bandwith is 1, so 2n or something steps would do it, and maybe, I don't know how many steps we have here. I guess, when we cut it in half that required us to do that much. It's order n, it's order n.

So the key question is, can we do it 2d? Can we do the same thing in 2d? So I want to follow that plan in two dimensions where now u will be a whole row at a time, so I'm doing block 2d, block 2d. So, can I write down the equations in block 2d for whole rows, so ui is the vector? So now this is the 2d problem, so it'll be minus the identity, where instead of instead of 1, I have to write the identity, ui minus 2 and 2k, right? Oh no, what is the middle -- what's on the -- so multiplying ui, ui minus 1. I wanted to just say the same thing, but I have to write down the -- this is going to be

n squared equations, n at a time, a whole row at a time, and what's on the diagonal of k2d? Not 2k. It's 2i, is it 2i plus k? Yeah. Yeah, k plus 2i. Isn't that what we have on the diagonal of k2d1 times ui minus 1 minus iui is equal to some -- that's a whole row at a time. These are all vectors with n components now, right? The minus i, the 2i, and the minus i are the second differences of one of rows; their difference is in the vertical direction, and this kui the second difference is along the row.

OK, so same equation at the next row. So the next row is minus i, ui minus 1, k plus 2ui, ui, because that's now the diagonal block minus iui plus 1 is fi plus 1, and it's just taking a second to write this one. This is k plus 2i, ui plus 1, minus i ui plus 2 is fi plus 2, OK. Exactly the same, but now a row at a time in 2d. So the same idea is going to work, right, what do I do? I want to cancel this, so I multiply that row by k plus 2i. Before I multiplied it by 2, but now I have to multiply it by k plus 2i times the row, the whole row. So when I do that this cancels, so I have minus this, this guy wasn't affected. And this will cancel, the k plus 2i will cancel this one, just as before. This will not be affected, ui plus 2, and I have fi and k plus 2i, fi plus one and fi plus 2. That should have been i minus 1, and this should have been i, and this should have been i plus 1, sorry, mis-labeled label the f's, but no big deal. The point is the left side, and the point is what's in that space. What goes in that space? Well, it's minus i, k plus 2i squared, minus i, so this is k plus 2i squared, which before was so easy, and then the minus i, and the minus i is the minus 2i is multiplying the ui. Yeah, that was [? my last i. ?]

OK, this is my matrix from odd even reduction. It's just natural to try the idea, and the idea works, but not so perfectly, because previously in 1d, that just gave us the answer 2. It was 4 minus 2, but now in 2d, we have a matrix, not surprising, but what we don't like is the fact that the bandwith is growing. k was tri-diagonal, but when we square it, it will have five diagonal, and when we repeat the odd even cycles. When we do it again. Those five diagonals will be nine diagonals, and onwards. So, I'm getting half-sized problems. All the odd-numbered rows in my square are eliminated, this just involves the even-numbered rows, but the matrix is not tri-diagonal anymore, it's growing in bandwith. So, and then you have to keep track of it, so the final conclusion is that this is a pretty good idea, but it's not quite

as good as this one. It's not as good as the FFT-based idea.

Also, if you look to see what is most efficient -- see the eigenvectors are still here, so I could do three steps of this and then go to Fourier, and that probably is about right. So, if you really wanted to polish off a fast Poisson solver, you could do maybe two steps or three of odd even cyclic reduction, but then your matrix is getting messy and you switch to the fast Poisson solver, so it's not quite Poisson anymore, because it's has a messier matrix, but it still has the same eigenvectors. As long as we stay with the matrix k, we know it's eigenvectors, and we know that they're sines and that they're quick to work with. OK. Anyway, there you go. That's a fast algorithm for the lecture before the spring break.

So after the break is, first of all discussion of projects. If your project could include page -- and you could maybe email the whole project to Mr. [? Cho ?]. Maybe also, could you email to me a sort of summary page that tells me what you did, so I'll save the summary pages, and I'll have for Mr. [? Cho ?] the complete project with printout and graph, as far as appropriate, and so we'll spend some time on that, and then move to the big topic of the rest of the course, which is solving optimization problems, minimization, maximization in many variables. OK, so have a good spring break and see you a week from Wednesday. Good.