PROFESSOR: This morning's Siam News. Well, it's an article on something that's not totally up to date because it's a method that was developed 50 years ago called alternating direction and it was exactly developed-- first you did an iteration in the x directions. Of course, we're tridiagonal along every row, we're one-dimensional. And then the second half-step is an iteration in the y direction. Again, tridiagonal, so that iteration's very fast. You flip-- alternating direction describes it perfectly. And that method is still used. It got developed in the oil reservoir simulation world. That is, Houston. And the article says the evolution of techniques for oil reservoir simulation has continued. And then mentions 3-- so it went from this alternating direction to line over relaxation, LSOR over relaxation and I only spoke briefly about that. What's that word line mean? That means that we're taking hole rows at once. Working with blocks and I'll add some notes about that today. So that was the next step, line SOR. Then to the Newton-Krylov schemes, that's what's coming. Conjugate [? gradients ?] with ILU type preconditions. So again, we're talking about preconditioners and one important choice is this incomplete LU. So notes on Gauss-Seidel and I should maybe add incomplete LU.

By the way, experiments badly needed here too. Because incomplete LU has this tolerance and as you move the tolerance you get closer or far away to the exact L and U, but you get faster or slower. So what's the balance? What's the right tolerance? How do these methods compare-- and overall, how do these iterative methods that we're speaking about compare with these direct methods? So we really are in this new chapter solving large systems, facing a whole lot of possible new experiments. So just to continue the history or the future as described in Siam News, so there was alternating direction, line SOR. Then to Krylov schemes with ILU preconditioners in use today. So this is a big use area and the multiscale, multigrid solvers that are quote, now starting to come of age. So that's a third topic

for today's lecture, multigrid. That of course, it's been around for some time, but it involved, especially at the start more effort in programming. So it didn't quickly-- I mean, it gave great results on model problems. On other problems, too, but still it kind of hasn't yet, but is penetrating into production code or the oil reservoir industry or other big industries would use it. So here you see really a pretty fair picture of where we are today with large solvers. These are very important.

Minimum degree-- if the size is not overwhelming direct methods are going to be right up there. And these are the pure iterations, stationary iterations that with this idea, this incomplete LU idea get pretty good. But the formula's always simple there. The new x comes from the old x by the same iterations. Whereas, if we use the ideas there as preconditioners and look to see-- so the point was, and I'll repeat it again, that Gauss-Seidel and Jacobi, the standard iterations were quite satisfactory on high frequencies. They damped those out quite well in the error, but they don't tackle the low frequency, the smooth part of the error, very efficiently. And that's what multigrid does incredibly well, so it's a nice mixture of the two. These are the smoothers, this is the change of scale, change of grid that's coming now and will come. So all this will be my subject for this week and next, prior to spring break and I think you will have enough to go on to do experiments. It could be highly interesting. OK, so that's the Siam News report.

OK, what do I want to say about minimum degree? And more will go on the web. Just a few more notes about what we did on Monday after class. If you take a matrix like that, you take a matrix and how is it described in sparse format? Sparse format you can see-- normally you don't have to see it, it's usually inside the sparse math lab, but if you want to see what's happening you could ask for i, j, and s. Now what are those? What will that produce? i will be the list of nonzeroes, the list of the row numbers of the nonzeroes. So i will be-- maybe I'll write it as a row so I'm transposing, So the row numbers would be row 1, row 2, not row 3. Row, 1, 2, 3, row 2, 3. The column numbers would be-- these are going to be the pairs i,j. So that's column 1 twice, column 2 three times. Column 3 twice. And what's s? s is the actual numbers in position i,j is the number 2. In position 2,1 is the number minus 1. So those numbers minus 1, 2, minus 1 is that column. And then minus 1, 2 is that

column. So of course, we've got all the information in the matrix. We know every nonzero position and we know what that entry is. And of course, our matrices are much bigger than this one, but already we can see one useful point. Point being that key word that the column numbers-- this I mean, imagine we have a large matrix, so this j is not very efficiently recorded here because what's j? We're looking at nonzeroes a column at a time, so of course, we'll have a few ones for the nonzeroes in column 1 and then some twos and then some threes and some fours, but all this row is-- the real information in that row is a pointer to-- in other words, I don't have to repeat 2 three times.

So really if I put inside here a compressed version of j, a pointer would be a short factor that just has a 1. It says look in position 1 for the start of column 1. Then a 3, see I didn't need that because that just continued column 1. That just repeated the j I already had, but this tells me that in the third position I start down column 2. And this would tell me that in position number 4, 5, 6, I guess, I start down column 3. You see the point of that 6. That 6 picks out-- yes-- 1, 2, 3, 4, 5, the sixth row number is the beginning of information on column 3. And then it's conventional to have a pointer, 8, to say finish. So that's pointing to empty space. So in other words, j got compressed to 1, 3, 6, 8 and in a large problem it would get seriously compressed. OK, so that's the form in which the code keeps the matrix and does the reoderings. Let me just mention that the opposite of this would create the matrix out of these i, j, s. What command would that be? You could create the matrix a out of the command sparse, would be good. Sparse of the inputs now would be i, j, s. If I input i, j, and s then that lab creates a matrix. And sometimes I may want to include the m and n, the shape of the matrix as further parameters, but actually here I wouldn't have to. So this is the opposite command from this one that we're interested in. You can imagine.

Suppose by using minimum degree or some other decision, I've eliminated up to a certain point and I want to put the remaining columns in a different order. I can do that just by playing with the pointers. So it's a very efficient structure for sparse matrices. And then comes the question, OK, what's the good order? I was surprised

to realize how open a problem that still is, even from the expert who's developing the key code. So approximate minimum degree of course by that word you see that it allows freedom. There are also decisions to be made when minimum degree is a tie between several nodes as it commonly is. And it's rather nice to get a math lab movie that shows the order in which nodes are eliminated, edges are removed from the graph, from the mesh. You'll see that. All right, I'm ready to go ahead now to some comment on Gauss-Seidel. Maybe I'll put that here. In fact, maybe I'll just take the same matrix. So now I want to remember, what is the Gauss-Seidel method for ax equal b. What's the iteration now? And again, a will be the same: 2, minus 1, zero minus 1, 2, minus 1. My favorite, OK.

So all these iterations you remember are splitting of the matrix. Some of the matrix goes on the left-hand side. That's the preconditioner. So it's p, the preconditioner that's sort of close to a in some sense, but easy to work with, multiplies the new thing and on the other side is p minus a-- the rest of the matrix-- that multiplies the old plus right-hand side b. Let's just remember again, that if we converge to the point where this is the same as this then we have the right answer. We have the right answer because if that's the same as that, px is the same as px, this comes over on the other side, ax equals b. So when it converges it converges to the right answer, but you remember that the key equation was that the new error is the old error multiplied by this matrix i minus p inverse a. That's the iteration matrix you could say. I just get it when I multiply both sides by p inverse. And so the problem is choose p so that this is easy to do and at the same time, this has small Eigen values or as small as you can get.

So the Gauss-Seidel is a particular choice, which takes the lower-- p is the lower triangular part here. So p is, so I just thought I'd better write down explicitly Gauss-Seidel takes that with zeroes there as multiplying-- that's p. And what's p minus a? It's the rest and it's been moved to the other side so the rest is going to be the strictly upper triangular part. And because it's moved over to the other side it'll have a plus sign. It'll be 1 and 1. x sub k plus b. I just write it out so that you see, totally explicitly how being triangular makes the solution step immediate. Because the first equation will tell us the first component, right? The first equation, because it's

triangular there's only an entry here. And by the way, all these codes, including Tim Davis' minimum degree codes, their very first step is reorder the equations if necessary to guarantee that the main diagonal has no zeroes. We want to know that in advance. So that you just do, let's assume of course, for us it happens automatically. OK, so that's not zero. And that first equation gives us the first component then we know the first component so we use it here and the second equation tells us what the second component is. We use that second component in the third equation to find the third component. So there's no loss of speed compared to diagonal and actually, it's faster because the storage we're changing-- we're using the new first component to find the second and the new second component to find the third, so we can overwrite x sub k by x sub k plus 1. Let's see. I guess if I had space and had prepared I would figure out what this matrix is. Maybe you could do that. Figure out what-- here's p. It's invertable. P inverse is going to be quite simple. Actually, it would be extremely simple. You'll be able to see its Eigen values immediately. We could stop to do it, but I think if you do it it's more valuable. So we would find that its Eigen values were below 1 and that they were the squares of the Jacobi Eigen values, so the method is twice as fast as Jacobi. But it has the same good feature of damping the high frequencies and the same bad feature of not very much damping the low, smooth part of the error.

Now we're ready for multigrid, which is intended to solve it. So multigrid is going to be-- the point is by changing grid, by changing from the fine grid to a coarse grid the smooth oscillation doesn't look so smooth. On the coarse grid, it's effective frequency is effectively doubled and it's moving over in the direction where the smoother can attack it. We'll see that happen.

So now, really this begins the lecture on multigrid, which is going to take certainly, today and Friday. There's a very good book by Briggs and others. He wrote the first edition of the book and coauthored the second one. And it's beautifully short and clear and simple and this presentation will follow the same exposition that the book does. So the key idea is to see what-- how to go from a fine grid problem, so this is on the fine grid with step size a. So this A sub h is our problem. A sub h is A. A sub

h, b sub h, we're looking for u sub h on the fine grid. Fine grid means lots of mesh points, lots of unknowns, big problem. OK, so the idea is going to be somehow to-- let me just start with two grids. Well, you see what you do on the fine grid at the start. You iterate. You use Gauss-Seidel or Jacobi, whatever. Maybe three times. Maybe three iterations, but don't continue to a thousand iterations, it's a waste of time. Then here is the real-- this is the multigrid part. The multigrid part is going-- you get an answer after a few iterations, after you've smoothed it. You compute this residual, which is the amount you're wrong in the equation, the difference between the right side and the left side. I can put h is here too, to emphasize. This is residuals being computed with what we have, which is on the fine grid.

Now here's the change of grid. Number two says, restrict to a coarse grid. Take that rh, that residual which is probably a bit smooth and move it to the coarse grid. So the coarse grid we identify by the fact that it's mesh width is twice as big, 2h. And so we're going to need some restriction, so the input to this multigrid is to decide on a restriction matrix, how shall we take values that are given on a fine grid and restrict them to a coarse grid. Of course, one restriction is-- and it's not that bad-- is just to take the values of r that are on the coarse grid at every other grid point, use those. Or we could take into account the neighbors. You see the question. So this is coming now in this restriction. So the restriction is, restriction r, this is fine to course, and we have this question of use neighbors or not? That will answer it. Well, the decision on what the matrix r is. That's the key.

OK, so we choose an r, we restrict, and we have now a half-size problem. Well, half-size in 1D, quarter-size in 2D, eighth-size in 3D because the mesh width has doubled in every coordinate. Now I've put solve. I better put quotes around that. Well, solve the problem on the coarse mesh. So the idea is that that's much faster Well, you might say, what problem? Well, we have to create somehow the matrix on the coarse mesh. That's not necessarily given to us. We start the problem with A, I mean, A sub h, the matrix on the fine. Can I just mention that this is probably, this v-cycle-- that letter v is supposed to suggest going down to the coarse mesh and back up to the fine mesh. And the standard notation in all the multigrid books is a capital V. And why do I use a small v? This is my educational contribution. Capital V is

appropriate when you have several meshes. You go to 2h, you go to 4h, you go to 8h, back to 4h, back to 2h, back to h. So a deeper multigrid. So it would look more complicated. I would repeat this idea instead of solve here. Instead of solve at the 2h level, which of course, I'm not going to do exactly anyway. But if I was in a big V-cycle I would iterate a few times to smooth here, just as I did here. I would iterate using weighted Jacobi or something and then go down to the 4h mesh. So restrict to the 4h mesh. So you can see that I would stay in this little loop to get to the bottom, 8h and then I would start back up. So I think it's a good idea to use a small leter v to tell us rather than saying two grid capital V-cycle. I'm just going to use a small v to signal right away that it's two grids. Fine, coarse, fine.

OK, so we have some solution, not necessarily exact and actually, there isn't that much point in getting it exact because what we want is to move toward the right answer. And notice that I'm looking here at the error rather than looking at A 2h, u 2h-- the actual solution. What I'm computing with is the correction term. So this is the correction term that I find. So this is the residual over here. this is the A 2h that I still have to choose and then solving that will give me a correction, but that correction is on coarse mesh. I've only had that correction E 2h defined on the 2h mesh. So now comes the other part of multigrid. Get back to the fine mesh, climb back up. I take that E 2h and apply an interpolation. I need to create an interpolation. This is going to be coarse to fine. And I have to decide how to do that. Once I've decided it that gives me a correction at the fine mesh level, which is where I'm really working, so I make the correction and I have a better answer. And probably, I iterate a few times-- that would be called a post smoother on that correction. And maybe that's where I stop or maybe I repeat little v-cycles, but that's not brilliant. If you're going to do multiple, if you're going to repeat multigrid it's much more efficient to move to more and more coarse meshes because the calculations on these are way faster than on the fine mesh.

You see, rather than-- I much prefer to go way down and back up then a lot of v-cycles. I mean, that would be a small w-cycle. And not brilliant. Much better to use a big V-cycle or a big W-cycle. There's a place for W-cycles but they're capital W-

cycles because you want to get down where it is very fast, very inexpensive and in fact, so efficient that multigrid achieves this holy grail of giving you an answer with whatever accuracy you want-- giving you an answer in o of n operations. n being the matrix size. Captial N squared in our example. So that's the fantastic result from multigrid. So I'll come back to this, but just say if the size of A is N, which is capital N in 1D, captial N squared, capital N cubed, then the multigrid works in o of in flops, Floating Point Operation. Not even log n, which we think of for the-- n log n we think of for the FFP, for the Fast Fourier Transform. But here's it's actually order of n. That really is a goal worth achieving.

So we know what the smoothers might be. It's the i and the r that are new. And then the analysis of convergence, why does it converge? But to use the method, and so by the way, projects using multigrid are totally welcome also. So let me take the interpolation i. So what's that? Coarse to fine. Let me imagine I'm in 1D, so there's the x direction. Here's one end of the interval, here's the other end of the interval and suppose my boundary conditions are zero. So this is the coarse mesh and let me put in here, make it the fine mesh. So at this point I have an answer on the coarse mesh, which satisfies the boundary condition and let's say it's there, there, and there. OK, and I need values on the fine mesh at the other points. Of course, I'm going to use these values when I interpolate. I mean, that word interpolation means implies, keep what you have on the given points and I'm just going to do linear interpolation. So halfway there, halfway there, and halfway there. So ignoring the boundary conditions, the zeroes, it's going to be 5 by 2 I think, this matrix i in this example will be 5 by 2 because it has 2 inputs, these and it has 5 outputs. And what's the matrix? Well, for their second and fourth outputs it uses exactly-- this multiplies x or I should say u, u sub 1 and u sub 2. Follow the notes again. The notes use v sub 1 and v sub 2 for the coarse mesh values to avoid h 2h, extra subscripts. So this height is v sub 1. This height is v sub 2 and I'm going to take-- this'll be the new u sub 1, u sub 3, and u sub 5 And u sub 3 I'm going to save. I better put i and make this into a true equation, i sub v. This is going to give me the values u sub 1, u sub 2, u sub 3, u sub 4 and u sub 5. So u sub 2 is just v sub 1. u sub 4 is just v sub 2. No problem. And what do I do with the other ones?

Let's see. I guess u sub 1 is halfway between v1 and 0. OK, so it's 1/2. u sub 1 is just 1/2 v1 It doesn't involve v2. u3, which is sitting here is 1/2 of these so it's 1/2 of this and 1/2 of that where this was nothing. Then u sub 4 was the same as v sub 2, just take v sub 2. And what's u sub 5? u sub 5 is 1/2 of v sub 2 and 1/2 of zero. So it's a simple matrix. The interpolation matrix. It's just reasonable to use the letter I and we all recognize here it's a rectangular matrix, not the identity. I doesn't stand for identity here, it stands for interpolation. OK, so I hope you can focus on the matrix. And if it was much, much bigger, but still in 1D it wouldn't look very different. Each column would have 1/2, 1, 1/2. That's a typical part of the interpolation matrix. It uses the value v sub 1, saves it at the center point, uses 1/2 of it at the previous point, and 1/2 of it at the right-hand point. And similarly, you see it's sort of a typical rectangular matrix in signal processing as well. I have to say something about what happens in 2D because our problems are in 2D. Maybe I just draw a typical 2D mesh. These are the coarse values, this is the coarse mesh and now of course, we'll save those values. I will have a 1, so it'll save those values when it goes to the fine mesh, the fine mesh being 1/2. So I just have to decide what to do there. So question, what value shall I take? Oh, I suppose I have to decide there, too. These are all new points and what shall I do. I'm just going to stay with linear interpolation, which is quite fast and satisfactory. This value will be the average of those two. This value will be the average of those and what will this one be in the center? It's the average of these four, of course. And that's what we'll get. Actually, that Kronecker product business, a Kronecker product of a 1D matrix like that with itself would be a 2D interpolation that would do exactly this. We can write it out more fully, but it's not fun to write out really fully because if this is 5 by 2 then I 2D would be 25 by 4. So we're right away, even on this tiny, tiny problem, we're right away up to 25 internal mesh points on the fine mesh. That's the matrix that I want you to see and the natural choice for R is to transpose.

So one possible choice-- and it will, the answer will be yes, we do use [? Naver. ?] So R will be the transpose of I. Actually, I'll need a factor here. I think it's 1/2. Let me just see what it is. Let me see why we need a little multiple there to make things right. So there's my matrix I, so I'm just going to transpose it and let me factor 1/2

out of that so you see these matrices without fractions inside. That would be 1, 2, 1. So I'm going to transpose that. 1, 2, 1, zero, zero. Zero, zero, 1, 2, 1. So I factored 1/2 out of I when I did that. And then I believe that I need another 1/2 here. I think I need 1/4. And what do I mean by needing 1/4? Of course, the main point is that this is the transpose of this. And you'll see that that preserves the symmetry that's just sort of the natural thing to pick. Not the only possibility, but it's very good to connect the restriction with the interpolation.

Of course, you can't take the restriction to be the inverse of the interpolation. Why not? I mean that sort of mental idea is that the restriction goes in one direction and the interpolation in the other direction. So why not just let one be the inverse of the other? Well of course, that matrix doesn't have an inverse. I mean it's rank is only 2. When we do a restriction we're going to lose information, the interpolation can't put it all back because we threw it away when we restrict it to the coarse mesh.

Now why 1/4? I think it's just if we had all constants, like all ones as our mesh values, we would want to get all ones for the restriction. So if I multiply this matrix by the vector of all ones-- so assuming you remember where R comes in by just copying this equation, you remember that when I go from the h mesh, it's fine to coarse. So R times ones on the fine mesh, all ones on the fine mesh gives ones on the coarse mesh. That's why I wanted that-- you see, if I multiply by all ones then that multiplication gives me 1 plus 2 plus 1 equal 4 and I need to divide by 4 to get back to the 1. You quickly discover that that just arises because of a change of scale. It's just a factor of 1/2. In 2D that 1/2 there would change to 1/4. In 3D it'd be 1/8. It's just the scaling to preserve constants.

And now the remaining question is, what's A 2h. That's the other matrix here that we're not given. We're given the A sub h, our problem. We choose an R and we choose an I and if we're smart, ones that transpose of the other. And then the question is what's A 2h? And there's a beautiful answer, so let me just say what it is. A 2h, a coarse mesh matrix is the fine mesh matrix, but I need to do first-- if this is going to apply to the coarse mesh guys, the v's I have to do an interpolation. You'll see it here. This is 5 by 5 in my problem. So I need to first do the interpolation to get

from a vector blank 2 up to a vector blank 5. Then this and then the restriction. That's terrific. That's A 2h. So this is A 2h and we'd better do an example. I guess it's going to be next time. So that'll be the first thing for next time, would be to see OK, for the standard second difference A sub h, for these piecewise linear interpolation and restriction, what comes out as A 2h? Do we get the standard second difference on the coarse mesh, which you really hope we do and we do. This will be the same difference in our example on the coarse mesh. So that this middle step of multigrid is exactly what we would have expected. What we woud've had if we set up the problem originally on the coarse mesh. But now we're going to get an answer that we take back to the fine mesh.

OK, so I'll do that next time. Some homeworks here that I held for an extra two days and now I'm returning and some thoughts to go up on to the website about possible projects, but actually, I expressed a lot of them at the beginning of the lecture. OK, thanks. Good.