NARRATOR: The following content is provided by MIT OpenCourseWare under a Creative Commons license. Additional information about our license and MIT OpenCourseWare in general is available at ocw.mit.edu.

PROFESSOR: So I'm thinking of large sparse matrices. So the point about a -- the kind of matrices that we've been writing down, maybe two dimensional, maybe three dimensional difference matrices. So it might be five or seven non 0s on a typical row and that means that you don't want to invert such a matrix. These are big matrices. The order could easily be a million. What you can do fast is multiply a by a vector, because multiplying a sparse matrix by a vector, you only have maybe five non 0s times the n -- the vector of length n, so say five n operations -- so that's fast. I'm thinking that we are at a size where elimination, even with a good ordering, is too expensive in time or in storage. So I'm going from direct elimination methods to iterative methods. Iterative means that I never actually get to the perfect answer x, but I get close and what I want to do is get close quickly.

So a lot of thought has gone into creating iterative methods. I'll write down one key word here. Part of the decision is to choose a good preconditioner. I'll call that matrix p. So it's a matrix that's supposed to be -- that we have some freedom to choose and it very much speeds up, so the decisions are, what preconditioner to choose? A preconditioner that's somehow like the matrix a, but hopefully a lot simpler. There are a bunch of iterative methods and that's what today's lecture is about.

Multigrid is an idea that just keeps developing. It really is producing answers to very large problems very fast, so that will be the following lectures and then come these methods -- you may not be familiar with that word Krylov. Let me mention that in this family, the best known method it is called conjugate gradience. The conjugate gradient, so I'll just write that down. Conjugate gradient, so that will come next week. That's a method that's a giant success for symmetric positive definite problem. So I don't assume in general that a is symmetric or positive definite. Often it is if it comes from a finite difference or finite element approximation. If it is, then conjugate

gradience is highly recommended.

I'll start with the general picture. Let me put the general picture of what an iteration looks like. So an iteration computes the new x. Let me call it x-new or xk plus 1 from the known x. So we start with x 0, any x 0. In principle, it's not too important for 0 to be close in linear methods. You could start even at 0. In nonlinear methods, Newton methods, you do want to be close and a lot of attention goes into a good start. Here the start isn't too important; it's the steps that you taking. Let me -- so I guess I'm going to -- I'm trying to solve ax equal b. I'm going to split that equation into -- I'm just going to write the same equation this way. I could write it as x equals i minus ax plus b. That would be the same equation, right? Because x and x cancel. ax comes over here and I have ax equal b. I've split up the equation and now actually, let me bring -- so that's without preconditioner. You don't see a matrix p. That's just an ordinary iteration, no preconditioner.

The preconditioner will come. I'll have p and it'll go here too. Then of course, I have to change that to a p. So that's still the same equation, right? px is on both sides, cancels -- and I have ax equal b. So that's the same equation, but it suggests the iteration that I want to analyze. On the right hand side is the known approximation. On the left side is the next approximation, xk plus 1. I multiply by p, so I -- the idea is p should be close to a. Of course, if it was exactly a -- if p equalled a, then this wouldn't be here and I would just be solving in one step ax equal b. So p equal a is one extreme. I mean, it's totally, perfectly preconditioned, but the point is that if p equals a, we don't want to solve that system. We're trying to escape from solving that system, but we're willing to solve a related system that is simpler. Why simpler?

Here are some possible preconditioners. p equals the diagonal. Just take the diagonal of a and that choice is named after Jacobi. That's Jacobi's method. Actually, it's already a good idea, because the effect of that is somehow to properly scaled the equations. When I had the identity here, I had no idea whether the identity and a are in the right scaling. a may be divided by delta x squared or something. It could be totally different units, but p -- if I take the diagonal of a, at least they're comparable. Then you see what -- we'll study Jacobi's method. So

that's a very simple choice. Takes no more work than the identity, really. Second choice would be -- so that would be p for Jacobi. A second choice that you would think of pretty fast is the diagonal and the lower triangular part. So it's the -- I'll say the triangular -- I'm going to say lower triangular, triangular part of a, including the diagonal. This is associated with Gauss's great name and Seidel. So that's the Gauss-Seidel choice. So why triangular? We'll analyze the effect of this choice. If p is triangular, then we can solve -- I'm never going to write out an inverse matrix. That's understood, but I can compute the new x from the old x 1 just by back substitution. I find the first component of x, k plus 1, then the second, then the third. It's just because the first equation -- if p is triangular, the first equation will only have one term, one new term on the left side and everything else will be over there. The second equation will have a diagonal and something that uses the number I just found for the first component and onward. So in other words, triangular matrices are good.

Then people thought about combinations of these. A big lot of activity went in something called overrelaxation, where you did a -- you had a weighting of these and it turned out to be if you adjusted the weight, you could speed up the method, sometimes called SOR-- successive overrelaxation. So when I was in graduate school, that was a very -- that was sort of the beginning of progress beyond Jacobi and Gauss-Seidel. Then after that came a very natural -- ilu is the next one that I want -- the i stands for incomplete, incomplete lu. So what are l and u? That's the letters I always use as a signal for the lower triangular and upper triangular factors of a, the ones that exact elimination would produce. But we don't want to do exact elimination. Why? Because non 0s fill in. Non 0s appear in those spaces where a itself is 0. So the factors are much -- have many more non 0s than the original a. They're not as sparse.

So the idea of incomplete lu is -- and I'll come back to it -- the idea of incomplete lu -- you might say, should have occurred to people earlier -- only keep the non 0s that are non 0s in the original a. Don't allow fill in or allow a certain amount of fill in. You could have a tolerance and you would include a non 0 if it was big enough, but the

many, many small non 0s that just fill in and make the elimination long, you throw them out. So p is -- in this case, p is l, is an approximate l times an approximate u. So it's close to a, but because we've refused some of the fill in, it's not exactly a. You have a tolerance there where if you set the tolerance high, then these are exact, but you've got all that fill in. If you set the tolerance to 0, you've got the other [? instance. ?]

So that would give you an idea of preconditions, of reasonable choices. Now I'm going to think about -- first, how do -- how to decide whether the xs approach the correct answer? I need an equation for the error and because my equations are linear, I'm just going to subtract the upper equation from the lower one and I'll call the -- so the error ek will be x minus the true answer. This is x exact. Minus x, my case approximation. So when I subtract that from that, I have ek plus 1 and here I have p minus a. I subtract that from that. I have ek and when I subtract that from that, 0. So very simple error equation. so this is an equation, this would be the error equation. You see the b has disappeared, because I'm only looking at differences now. Let me write that over on this board, even slightly differently.

Now I will multiply through by p inverse. That really shows it just so clearly. So the new error is, if I multiply by p inverse, p inverse times p is i and I have p inverse aek. So what does that say? That says that at every iteration, I multiply the error by that matrix. Of course, if p equals a, if I pick the perfect preconditioner, then p inverse a is the identity. This is the 0 matrix and the error in the first step is 0. I'm solving exactly. I don't plan to do that. I plan to have a p that's close to a, so in some way, this should be close to i. This whole matrix should be close to 0, close in some sense. Let me give a name for that matrix. Let 's call that matrix m. So that's the iteration matrix, which we're never going to display. I mean, it would be madness to display. We don't want to compute p inverse explicitly or display it, but to understand it -- to understand convergence or not covergence, it's m that controls everything.

So this is what I would call pure iteration or maybe another word that's used instead of pure is stationary iteration. What does stationary mean? It means that you do the same thing all the time. At every step, you just use the same equation, where these

methods will be smarter. They'll adapt. They'll use more information. They'll converge faster, but this is the simple one -- and this one plays a part in those.

So what's the story on convergence now? What about that matrix m? If I take a starting vector, e 0 -- my initial error could be anything -- I multiply by m to get e 1. Then I multiply m by m again to get e 2. Everu step, I multiply by m. So after k steps, I've multiplied k times by m. I want to know -- so the key question is, does that go to 0? So the question, does it go to 0 and if it does, how fast? The two parts, does it go to 0 and how fast are pretty much controlled by the same property of m. So what's the answer? When does -- if I take -- I don't know anything about e 0, so what property of m is going to decide whether its powers get small, go to 0, so that the error goes to 0? That's convergence. Maybe wall up or maybe stay away from 0. What controls it?

One word is the answer. The eigenvalues. It's the eigenvalues of m, and in particular, it's the biggest. If I have an eigenvalue of m, as far as I know, e-not could be the eigenvector and then every step would multiply by that eigenvalue lambda. So the condition is all eigenvalues of m have to be smaller than 1 and in magnitude of course, because they could be negative, they could be complex. So that's the total condition, that's exactly the requirement on m. How do we say how fast they go to 0? How fast -- if all the eigenvalues are below 1, then the slowest convergence is going to come for the eigenvalue that's the closest to 1. So really it will be -- and this is called the spectral radius and it's usually denoted by a Greek row of m, which is the maximum of these guys. It's the max of the eigenvalues and that has to be below 1. So it's this number, that number, because it's that number which really says what I'm multiplying by it every step. Most likely, e-not, my initial unknown error has some component in the direction of this biggest eigenvalue, in the direction of its eigenvector. Then that component will multiply at every step by lambda, but that one will be the biggest guy, rho. So it's the maximum eigenvalue.

That's really what it comes to. What is the maximum eigenvalue? Let me take Jacobi. Can we figure out the maximum eigenvalue for Jacobi? Of course, if I don't know anything about the matrix, I certainly don't want to compute eigenvalues. I

could run Jacobi's method. So again, Jacobi's method, I'm taking p to be the diagonal part, but let me make that explicit and let me take the matrix that we know the best. I'll call it k. So k or a -- this is the a -- is my friend with 2s on the diagonal minus 1s above. I apologize -- I don't really apologize, but I'll pretend to apologize for bringing this matrix in so often. Do we remember its eigenvalues? We computed them last semester, but that's another world. I'll say what they are. The eigenvalues of a -- this is an m now -- the eigenvalues of this matrix a -- I see 2s on the diagonal. So that's a 2. That accounts for the diagonal part, 2 times the identity. I have a minus and then the part that comes from these 1s, which are forward and back.

What would von Neumann say? What would von Neumann do with that matrix? Of course, he would tested on exponentials and he would get 2 minus e to the ik minus e to the minus ik. He would put the two exponentials together into cosines and that's the answer. It's 2 minus 2 cosines of whatever angel's theta. Those are the eigenvalues. The j'th eigenvalue is -- the cosine is at some angle theta j. It's worth since -- maybe just to take again a minute on this matrix. So the eigenvalues are between 0 and 4. The eigenvalues never go negative, right? Because the cosine can't be bigger than 1. Actually, for this matrix the cosine doesn't reach 1. So the smallest eigenvalue is 2 minus 2 times that cosine close to 1. It's too close for comfort, but it is below 1. The eigenvalues are positive and they're between 0 and 4. At the other extreme, theta is near pi. The cosine is near minus 1 and we have something near 4. So the eigenvalues of that -- if you look at that matrix, you should see. Special matrix eigenvalues in this interval 0 to 4. The key point is, how close to 0?

Now what about m -- what about p first? What's p? For Jacobi, so p for Jacobi is the diagonal part, which is exactly 2 i. That's a very simple diagonal part, very simple p. It produces the matrix m. You remember m is the identity matrix minus p inverse a. That's beautiful. So p is just 2 i. So I'm just dividing a by 2, which puts a 1 there and then subtracting from the identity. So it has 0s on the diagonal. That's what we expect. It will be 0s on the diagonal and off the diagonal, what do I have? Minus -- with that minus, p is 2 so it's 1/2. It's 1/2 off the diagonal and otherwise, all 0s.

That's the nice matrix that we get for m in one dimension.

I have to say, of course, as you know, we wouldn't be doing any of this for this one dimensional matrix a. It's tridiagonal. We can't ask for more than that. Direct elimination would be top speed, but the 2D case is what we understand that it has a big bandwidth because it's two dimensional, three dimensional even bigger -- and the eigenvalues will come directly from the eigenvalues of this 1D case. So if we understand the 1D case, we'll knock out the two and three dimensional easily. So I'll stay with this one dimensional case, where the matrix m -- and I'll even maybe shouldn't write down exactly what the iteration does -- but if I'm looking now at convergence, do I have or don't I have convergence and how fast? What are the eigenvalues and what's the biggest one? The eigenvalues of a are this and p is just -- p inverse is just 1/2 times the identity. So what are the eigenvalues? Again, I mean, I know the eigenvalues of a, so I divide by 2. Let me write down what I'm going to get now for the eigenvalues of m, remembering this connection. So the eigenvalues of m are 1 minus 1/2 times the eigenvalues of a. Better write that down. 1 minus 1/2 times the eigenvalues of a. Of course, it's very special that we have this terrific example matrix where we know the eigenvalues and really can see what's happening and we know the matrix.

So what happens? I take half of that -- that makes it a 1. When I subtract it from that one, it's gone. 1/2 of that, that 2 is cancelled -- I just get cos theta. Theta sub whatever that angle was. It's a cosine and it's less than 1. So convergence is going to happen and convergence is going to be slow or fast according as this is very near 1 or not -- and unfortunately, it is pretty near 1, because the first, the largest one -- I could tell you what the angles are. That's -- to complete this information, I should have put in what -- and von Neumann got them right. That's j -- is there maybe a pi over n plus 1? I think so. Those are the angles that come in, just the equally spaced angles in the finite case. So we have n eigenvalue. The biggest one is going to be when j is 1. So the biggest one, rho, the maximum of these guys, of these lambdas will be the cosine of when j is 1 pi over n plus 1. That's the one that's slowing us down. Notice the eigenvector that's slowing us down, because it's the eigenvector

that goes with that eigenvalue that's the biggest eigenvalue that's going to hang around the longest, decay the slowest. That eigenvector is very smooth. It's the low frequency. It's frequency 1. So this is j equal to 1. This is the low frequency, j equal 1, low frequency. The eigenvector is just samples of the sine, of a discrete sine. One lowest frequency that the grid can sustain.

Why do I emphasize which eigenvector it is? Because when you know which eigenvector is slowing you down -- and here it's the eigenvector you would think would be the simplest one to handle -- this is the easiest eigenvector, higher frequency eigenvector, have eigenvalues that start dropping. Cosine of 2 pi. Cosine 3 pi over n plus 1 is going to be further away from 1. Now I have to admit that the very highest frequency, when j is capital n, so when j is capital n, I have to admit it happens to come back with a minus sign, but the magnitude is still just as bad. The cosine of n pi over n plus 1 is just the negative of that one. So actually we have two eigenvalues that are both hitting the spectral radius and both of them are the worst, the slowest converging eigenvectors. This eigenvector looks very smooth. The eigenvector for that high frequency one is the fastest oscillating. If I graph the eigenvectors, eigenvalues, I will. I'll graph the eigenvalues. You'll see that it's the first and the last that are nearest in magnitude to 1.

I was just going to say, can I anticipate multigrid for a minute? So multigrid does -- is going to try to get these guys to go faster. Now this one can be handled, you'll see, by just a simple adjustment of Jacobi. I just wait Jacobi a little bit. Instead of 2 i, I take 3 i or 4 i. 3 i would be very good for the preconditioner. That will have the effect on this high frequency one that'll be well down now, below 1. This will still be the slowpoke in converging. That's where multigrid says, that's low frequency. That's too low. That's not converging quickly. Take a different grid on which that same thing looks higher frequency. We'll see that tomorrow.

So if I stay with Jacobi, what have I learned? I've learned that this is rho. For this matrix, the spectral radius is that number and let's just see how big it is. How big is -- that's a cosine of a small number. So the cosine of a small number, theta near -- theta near 0, the cosine of theta is 1 minus -- so it's below 1, of course -- 1/2 theta

squared. Theta is pi over n plus 1 squared. All these simple estimates really tell you what happens when you do Jacobi iteration. You'll see it on the output from -- I mean, you can -- I hope will -- code up Jacobi's method, taking p to be a multiple of the identity, so very fast and you'll see the convergence rate you can graph.

Here's an interesting point and I'll make it again. What should you graph in seeing -- to display convergence and rate of convergence? Let me draw a possible graph here. Let me draw two possible graphs. I could graph -- this is k as I take more steps. I could graph the error, the size of the error. It's a vector, so I take its length. I don't know where I start -- at e-not. This is k equals 0. This is the initial guess. If I use -- I'm going to draw a graph, which you'll draw much better by actually getting MATLAB to do it. Before I draw it, let me mention the other graph we could draw. This tells us the error in -- this is the error in the solution. If I think that's the natural thing to draw, it is, but we could also draw the error in the equation, the residual error. So I'll just put up here what I mean by that. If I put it in -- so ax equals b exactly. axk is close. The residual is -- let me call it r, as everybody does -- is the difference ax minus axk. It's how close we are to b, so it's b minus axk. The true ax gets b exactly right. The wrong ax gets b nearly right. So you see, because we have this linear problem, it's a times the error. x minus xk is ek. So this is the residual and I could grapj that. That's how close my answer comes to getting b right, how close the equation is. Where this is how close the xk is. So that would be the other thing I could graph. rk is the size of the residual, which is aek.

So now I have to remember what I think these graphs look like, but of course, it's a little absurd for me to draw them when -- I think what we see is that the residual drops pretty fast. The residual drops pretty fast and keeps going. Let's say Jacobi's method or in a minute or maybe the next time, another iterative method -- but the error in the solution starts down fast and what's happening here is the high frequency components are getting killed, but those low frequency components are not disappearing, because we figured out that they give the spectral radius the largest eigenvalue. So somehow it slopes off and it's -- the price of such a simple method is that you don't get great convergence.

Of course, if I put this on a log-log plot, the slope would be exactly connected to the spectral radius. That's the rate of decay. It's the -- ek is approximately -- the size of ek is approximately that worst eigenvalue to the kth power. Do you see how this can happen? You see, this can be quite small, the residual quite small. I think it's very important to see the two different quantities. The residual can be quite small, but then from the residual, if I wanted to get back to the error, I'd have to multiply by a inverse and a inverse is not small. Our matrix a is pretty -- has eigenvalues close to 0. So a inverse is pretty big. a inverse times aek -- this is small but a inverse picks up those smallest eigenvalues of a -- picks up those low frequencies and it's bigger, slower to go to 0 as we see in this picture. So it's this picture that is our problem. That shows where Jacobi is not good enough.

So what more to say about Jacobi? For this matrix k, we know its eigenvalues. The eigenvalues of m, we know exactly. The matrix m we know exactly. You see in that matrix -- now if I said, look at that matrix and tell me something about its eigenvalues, what would you say? You would say, it's symmetric, so the eigenvalues are real and you would say that every eigenvalue is smaller than 1. Why? Because every eigenvalue is -- if you like to remember this business of silly circles, every eigenvalue is in a circle centered at this number - in other words, centered at 0 and its radius is the sum of those, which is 1. So we know that every eigenvalue is in the unit circle and actually, it comes inside the unit circle, but not very much -- only by 1 over n squared. What do I learn from this 1 over n squared? How many iterations do I have to take to reduce the error by, let's say, 10 or e or whatever? If the eigenvalues are 1 minus a constant over n squared, I'll have to take the n squared power. So if I have a matrix of order 100, the number of iterations I'm going to need is 100 squared, 10,000 steps. Every step is fast, especially with the Jacobi choice. Here's my iteration -- and p is just a multiple of the identity here. So every step is certainly fast. Every step involves a matrix multiplication and that's why I began the lecture by saying, matrix multiplications are fast. a times xk -- that's the work and it's not much. It's maybe five non 0s and a -- so five n. That's fast, but if I have to do it 10,000 times just to reduce the error by some constant factor that's not good. So that's why people think Jacobi gives us the

idea. It does knock off the high frequencies quite quickly, but it leaves those low frequencies.

I was going to mention weighed Jacobi. Weighted Jacobi is to take instead of p -- and put in a weight. So weighted Jacobi -- let me put this here and we'll come back to it. So weighted Jacobi simply takes p to be the diagonal over a factor omega. For example, omega equals -- 2/3 is a good choice. Let me draw the -- you can imagine that if I -- this is 2 times the identity, so it's just 2 over omega i -- I'm just choosing a different constant. I'm just choosing a different constant in p and has a simple affect on n. I'll draw the pictures. The eigenvalues of m are these cosines. So I'm going to draw those cosines, the eigenvalues of m. So they start -- I'll just draw the picture of cos theta if I can. What does a graph of cos theta look like? From 0 to pi, so I'm going to see -- this is theta equals 0. This is theta equal pi. If I just draw cos theta -- please tell me if this isn't it. Is it something like that? Now where are these theta 1. the bad one. Theta 2, theta 3, theta 4, theta five, equally bad down here.

So there's cos, there's the biggest one. That's rho and you see it's pretty near 1. Here I'm going to hit -- at this frequency, I'm going to hit minus rho. It'll be, again, near 1. So the spectral radius is that, that's rho. The cosine of that smallest angle, theta over n plus 1. What happens when I bring in these weights? It turns out that now the eigenvalues of m with the weights will be 1 -- it turns out 1 minus omega plus omega cos theta. You'll see it in the notes, no problem. So it's simply influenced by omega. If omega is 2/3 -- if omega is 2/3, then I have 1 minus omega as 1/3 plus 2/3 cos theta. Instead of graphing cos theta, which I've done -- let me draw -- let me complete this graph for cos theta. The eigenvalues unweighted with weight omega equal 1. Now it's this thing I draw. So what happens? What's going on here? For a small theta, cosine is near 1. I'm again near 1. In fact, I'm probably even little worse, probably up here. I think it goes down like that. So it starts at near 1, but it ends when theta is pi. This is 1/3 minus 2/3. It ends at minus 1/3. That's a lot smarter. This one ended at minus 1, that one ended at near minus 1 but this one will end with omega -- this is the omega equal 2/3 of weighted 1. All you've done is fix the high frequency. When I say, that was a good thing to do, the high frequencies are no longer a problem. The low frequencies still are. So the only way we'll get out

of that problem is moving to multigrid.

Can I, in the remaining minute, report on Gauss-Seidel and then I'll come back to it? But quick report on Gauss-Seidel. So what's the difference in Gauss-Seidel? What does Gauss-Seidel do? It uses the latest information. As soon as you compute an x, you use it. Jacobi compueted all -- had to key all these xs until it found all the new xs, but the Gauss-Seidel idea is, as soon as you have a better x, put it in the system. So the storage is cut in half, because you're not saving a big old vector while you compute the new one. I'll write down Gauss-Seidel again. So it's more up to date and the effect is that the Jacobi eigenvalues get squared. So you're squaring numbers that are less than 1. So Gauss-Seidel, this gives the Jacobi eigenvalues squared. The result is that if I draw the same curve for Gauss-Seidel, I'll be below, right? Essentially a Gauss-Seidel step is worth two Jacobi steps. Eigenvalues get squared as they would do in two Jacobi steps. If I do Jacobi twice, I square its eigenvalues. So it seems like absolutely a smart move. It's not brilliant, because this becomes cosine squared. I lose the 1/2, but I'm still very, very close to 1. It still takes order n squared iterations to get anywhere. so Jacobi is, you could say, replaced by Gauss-Seidel as being one step better, but it's not good enough and that's why the subject kept going. This gives rho as 1 minus order of 1 over n first power, where these were 1 minus 1 over n squared. So this is definitely further from 1 and then this can be way further from 1.

So you can see some of the numerical experiments and analysis that's coming. I'll see you Wednesday to finish this and move into multigrid, which uses these guys.