PROFESSOR: This is my lecture on section 6.1 of the notes. Really, the central problem of numerical linear algebra is solving ax equal b by elimination. That's what is used much more often than anything else, but we're talking about sparse matrices, large sparse matrices. That's what we have in mind for a long time now. Probably all lectures coming up, thinking of those matrices and this is the prototype of a large sparse matrix.

The main point of the lecture is, if you take the unknowns in whatever order they're given -- here I took them numbering along the rows -- one, two, three, four, five, six, seven, eight and so on -- you got to stop and think or get the computer to look at reorderings, because the reorderings can make an enormous difference in the fill in. So this is -- actually, the key words for this lecture is and the key concept is the fact that as elimination goes forward, a lot of these 0s become non 0s. Let me see it happen, first of all. So this is the whole idea of fill in. What's going to be the very first entries that are filled in?

Let's see. I guess that the very first row operation, if you took the matrix like this and did elimination, 4 is the first pivot. You would add 1/4 of the first row to the second row to make that a 0, right? Then what would happen? This would change that number, but this would put a new number in there. So a new number would appear when we do that first row operation and then also working with this pivot, I would operate on that row to make this a 0 and then this would change, that would fill in, right? I guess this'll already has something to it. That won't be new. So each of those steps, each of those row operations filled in a non 0 and reduced the sparsity of the matrix, and the sparsity is what we're living by.

Now if I continue and continue, we'd need MATLAB, to use spye in MATLAB to see what the picture would look like. It's very interesting to see it. Spye shows you the non 0s. We're very much focused on what's 0 and what isn't 0 in matrix entries. We actually keep track of that, the structure of a matrix, as well, of course, as separately keeping track of the actual numbers. One place, we're keeping the actual numbers, but in the decisions on ordering, we're only interested in 0s and non 0s.

So let me look even more at this fill in. Now you saw it happen up there. Let me just look at a single fill in step. So I have a row j and a row i -- and this is going to be the pivot that I'll use to eliminate this, but in this row, there's another non 0 and in this row, the case entry is not non 0. It's 0. So this is what we start with prior to elimination, just as we did there. I just want to see focus on small on the fill in occurring. Of course, this isn't on the diagonal. This is in column k, which is somewhere far out. In that case, some are close by. In that case -- but we're just -- this is on the diagonal. That's the pivot, the j'th pivot, that I'm ready to use, so I subtract a multiple of that to remove that. So what is elimination -- will not change the pivot row and it will produce a 0 there, that's the point.

Then the real point that I'm making is that when I multiple this by the right number to produce 0 and do this subtraction, this 0 disappears, that fills in. That's some aik new that -- the new value in row i, column k is not 0. so we see that as we saw it for numbers. Here we're seeing it symbolically. Let me see it graphically, because everybody sort of -- to visualize the big picture. I think everybody finds the graph description of the non 0 structure the simplest to imagine.

So the graph for this little problem would be -- so what's the graph that tells me where the non 0s are? Again, it doesn't tell me what numbers they are, just tells me where they are, which positions. It's only the positions that I worry about in deciding on the ordering to use. Then I execute using the actual numbers, but in deciding on the ordering, it's just the position. So I think of my graph as having a node for every row. So that's node j for row j. This is node i for row i. Somewhere we'll have a node k. So at the start, so those are the nodes.

What are the edges? There is a non, so the edges come from the non 0s in the matrix. So there's an edge from j to k, because this is not 0. There's an edge from i to j or j to i. I'm not worrying about the direct -- this is not a directed graph right now. So this is the start, but there's no edge here initially, because aik is 0. Now in this graph world -- it's so small, we can just focus on the part that's going to change a little. The rest of all the other rows have their nodes and all the other non 0 entries have their edges. So this is a tiny part of the big graph.

What happens to this tiny part? So again, I want to see the same elimination step, just this way in the graph world. So after, what edges are there? So again, it's j, i and k -- and after I do elimination, j is still connected to k. That's still there. No change. j is no longer connected to y, because elimination killed that edge, but elimination created a new edge, i to k. So that's essentially what elimination does. Maybe we can say it in words. When this edge disappears, at the same time we create new edges from connecting any two nodes that were both connected to j. You see that point? If I'm using j as the pivot row and I look on that row and say, what non 0s are there? What edges are connected to j and I see one -- and then, when I do elimination, I'm going to introduce, exactly as has happened here, a new edge between any pair that were both connected to j. So we're kind of fighting against creeping filling.

The question is, what order should we do these eliminations? So now think of a larger graph. Actually, I guess we could somehow think of that -- we can try to visualize it, but maybe not succeed. So the graph for that problem -- could that seve as the graph? I guess it could, really. I would remove -- I'd [? whiz ?] away the -- if I'm given the values on the outer edges, then I would remove all those, but essentially this is the graph for that matrix, right? If I number each of the nodes 1 to 16 and I would do the numbering the way I did to produce that matrix -- now the whole question is, if I do some different numbering, will -- that'll shift around. That'll just be a permutation matrix, right? If I do a different numbering -- so matrix wise, I have my matrix k to d -- what happens matrix wise for -- if I do a numbering, a different numbering, a reordering? That just permutes the unknowns and the equations.

So really, what happens is some permutation multiplies k to d on the left and its transpose does the same thing to the columns on the right, because this permutation will change the order of the equations. This permutation will reorder the columns to come back, to bring the diagonal back onto the diagonal, but in a different order. We won't know it with all those 4s, but that will have, again, all the 4s on the diagonal, but the minus 1s are moved around. Of course, none have disappeared or been created by that. It's just that's giving us a new order. So that the reodering matrix wise What's the reordering graph wise? Graph wise would be, instead of this numbering of unknowns -- one, two, three, four, five, six, seven, eight and so on -- nine, ten, eleven, twelve, thirteen, fourteen, fifteen, sixteen -- we have a different order, different numbering. The computations, the computer science aspect, you could say, was not going to be difficult.

Actually, we should think and maybe we'll think in a minute, from the computer science aspect, the actual execution -- I'm probably not -- certainly not going to write out a permutation matrix. That's ridiculous. I'm not going to draw a graph and keep that -- I'll keep some list of positions for non 0s, so we'll say a word about that list later. Here I'm trying to say, how do we get an idea of what effective reordering is and what a good reordering would be? So here, this is a specific reordering that I want to mention right away, because it's so straightforward. You can get the idea of the red-black ordering without any big computations. This is the -- fundamentally, the winner. This is what people use all the time and MATLAB's backslash is going to have that available.

Inside this, the codes that are called are the column or symmetric matrix, approximate minimum degree. So that's what this AMD means -- approximate minimum degree. So that's the basic idea that I'll come back to as soon as I've just drawn the red-black picture. It's so important that people don't stop thinking. so another sort of high level picture of how to do an ordering is look for graph separators. That's become a big step in the world of computational graph theory, and that leads to something called nested dissection, which is separators of separators of separators, the usual keep going with the same idea that you see in algorithms.

What about red-black? Red-black, I'm simply going to imagine that this is a checkerboard -- it's the nodes that I'm looking at, not the squares. So that's red, black, red, black. So I'm going to number first -- I'm going to take all the reds first, so I'm changing this number now to take all the reds -- one, two, three, four, five, six, seven, eight and now all the blacks -- nine, ten, eleven, twelve and so on. I'll stop there so your eye will know how to fill in. What's the point of that? If I take a red node and look -- suppose this first one is a red node, as it is -- then what do I know about the positions of those minus 1s, the non 0s in the matrix? They're associated with black, right? The neighbors of this node number one are black. The black ones have numbers from nine to sixteen. So then the black -- everybody sees it. With this molecule that has a 4 in the center and minus 1s beside it, those minus 1s are associated with nodes of the opposite color. That's what we see here in the actual numbering.

So what does the matrix become? What does this -- if I do that permutation, the permutation that's associated with that renumbering -- what will the same matrix -- here's the same matrix, but now I'm permuting it and so what does it look like? Maybe I can draw it smaller. I think I probably can. So in this pretty special situation where the red nodes are not connected to any other red nodes, only to themselves, I think I'll only see the 4 a -- this is red to red. Those are the eight red nodes, an eight by right identity matrix times 4. The blacks are only connected to the black and all the minus 1s are in our connections between red and black. This far off the diagonal, what shall I call it? b, I'll just pick a letter -- b and transposed probably, because I know the matrix is going to stay symmetric. If it started symmetric, as it did, then this operation of reordering rows and columns compatibly is going to leave it symmetric and it must look something like this.

Is that matrix more sparse than before? No. It's got the same minus 1s. They're just all pushed up into this corner. Will it have some fill in as we go forward? What's the answer on that one? As I go forward, maybe I should -- I haven't really thought that through. My instinct is yes. We'll have fill in. Will we have more fill in or less fill in than the row by row ordering? I should know the answer. I think I'd probably ask the machine to tell me. My guess is probably not more and we can -- why do I think probably this is better? Of course, I should know. In general, it's better to put off the evil day -- that is, this goes through the first half of elimination -- I mean, you know what I'm thinking here. All the off diagonals have been sort of pushed down in the matrix. Let me do an example that shows why that's good. Suppose my matrix has a non 0 diagonal -- and by the way, the first step in the good codes for elimination -- do a permutation exactly to produce a non 0 diagonal, so you're OK to start. You know where you are.

Now suppose that all non 0s all show up and I'll make the matrix symmetric. That's the non 0 pattern and my point is, that's a very good non 0 pattern. Why is it good? It's good because there's no fill in whatever. When I subtract some multiple, when I go to make this 0, to eliminate it, I subtract a multiple using the first pivot. It changes that, but it doesn't fill in anything new. The only operations required are operating on the last row, which is already full. So there's an example in which the bandwidth -- see, normally the bandwidth is a good guide -- but that's only for pretty full bands.

Here's a matrix that's not banded, really. Its bandwidth is the full five. I have to go out five columns until I get the last, until I get the non 0s. So that is a big bandwidth. So bandwidth w, the bandwidth w is large, but it's the perfect ordering, but no fill. I don't fill anything -- you realize that for banded matrix, we won't have fill in outside the band. Our concern is for fill in inside the band and here, we don't have any, because the bad row -- and actually, it often happens in some b squares in another problem that you might -- this isn't quite as stupid a matrix as you might think. These might be blocks rather than single entries. You often want to think block wise about a matrix. You might think, why would you have a matrix that was well separated with block diagonal, which is really good, and then a few rows that are full.

That seems sort of unlikely, but I guess I want to say it isn't. If you had a problem in these squares minimizing some expression involving that matrix, the block diagonal part, but with the constraint that maybe the sum of all the unknowns has to be 1, then when you use a Lagrange multiplier to bring that constraint into the problem, into the matrix, that's going to show like that. and the beauty of the Lagrange multiplier device is that also that it'll produce a symmetric column there. We'll see that in the third part of the course, but I'm just -- my point is that this example is not as artificial as it seems.

Now suppose we had taken the bandwidth as our metric, as our goal. In reducing the bandwidth as our goal, we would have made a bad mistake. If I wanted to do a permutation that reduces the bandwidth, I better put this up in the middle somewhere. So this is -- that bad row comes up in the middle and then, I guess, these are maybe whatever -- I guess I need three xs down here. I'm going to do the same permutation from the right, so that will move this column over to the middle. This would be a reordered -- this would be a p matrix, p transposed thing, I think. It's got a better bandwidth, right? Now we have -- we're totally confident of all these 0s here. But it will fill in.

Where will we get fill in, in that matrix? So as we start, we're using elimination to remove those entries, no problem, because that upper left corner is the way this big, the hold matrix was, no fill in up to there, but just as soon as we reach this one as our pivot row and use it to subtract multiples to knock out those, when I make those 0 by using that pivot, it's going to bring the whole -- this will be gone at that time, but this will be non 0 and that will produce new edges in the graph. Might be fun to look at the graph that is associated with this matrix and its permutations. But anyway, you see that this will all fill in.

So again, in that small problem, we could really see what was happening. In the big problem, we can almost see it when we take a very specific, highly structured permutation like red-black, although I haven't pinned down exactly what the count is like now. I'd be glad if somebody would just run a computer experiment, count non 0s.

So what's the good thing to counts? How do you see what the fill in finally was when you've done an elimination? I guess that the lnu factors, which is what elimination produces, that's where the fill in will be. So maybe I just make that point, so that you could make these experiments that I have in mind and tell me what I ought to know about it. So if I have the matrix k, whichever it is -- whether it's this one or this one or any other, then the effect of elimination is the factor into a lower times an upper triangular matrix.

I'll just mention a special case while I'm -- special positive definite case -- we can adjust it so that's k is l times l transposed. This is called Cholesky. We can arrange the -- it will happen. We can arrange the scaling so that by putting the square root of pivots with l and the other square root with l traposed -- so Cholesky is a MATLAB command, chol and actually, MATLAB -- if you give MATLAB a symmetric matrix -- if you give MATLAB a matrix to work with, it will first ask -- it will ask, is it symmetric? If it is symmetric, it will hope that it's positive definitely, so it will begin Cholesky. It'll begin to look for this -- it'll begin the factorization, which is perfect for positive definite 1 until it succeeds and finishes Cholesky or until it discovers that a negative pivot appeared, which means with signals, that the matrix is not and was not originally positive definite and then MATLAB has to accept, OK, now I may have to do row exchanges, pivoting operations and all the rest. So MATLAB's sort of optimistic until it -- on a symmetric matrix until it is told otherwise.

But not all matrices are symmetric, so this would be the general case, almost the general case. I haven't included here the pivoting, the row exchanges that MATLAB does as it executes elimination. So let me just finish my sentence on l and u. I will have the fill in, in it. l will tell us and u will, certainly -- I mean, they both will. u will have these -- that guy is going to be part of u and this one is -- I expect to see something there in l. So l and u will actually exhibit the fill in, so that you could find out how much fill in there is simply by counting the non 0s in the matrices l and u.

This is to encourage you, because this is really a pretty experimental subject. Nobody -- the best methods, which -- or the favorite method. I think nobody can say best, but this approximate minimum degree, the favorite method, we could prove some theorem to show that it was better than something else, but to really see, is it successful or not on a wide variety of problems, that's really experimental. Just try it.

Now what does it mean? So I looked at that specific one. Now I'm ready to look at the approximate minimum degree and I'll contrast it with the perfect minimum degree. There are codes for that and those as become ms. But it turns out -- and I don't know exactly why it's the case -- that the approximate minimum degree usually, if anything, it's superior. So these are the ones to use. So what does minimum degree mean? Minimum degree means -- let's see -- if I took this problem -- I should draw the graph for this matrix. So what does the graph look like for this matrix? I've got six nodes -- let me put six nodes in -- let me put five and the node in the middle will be the one that they're all connected to. In this ordering, those would be one, two, three, four, five and six -- and all the connections go into six. So that's the graph for that matrix.

The graph for this one -- so you probably see it better than I do. What I had originally there -- I guess this is the graph with the numbering one, two, three, four, five, six -- what was there before the fill in happened -- the graph would look the same, but the nodes would have a different numbering, right? One, two, three, four, five, six -- so shall I just put the bad guy here? One, two, three, four, five, six and then what's the point? As soon as I use number three to eliminate four -- three was connected to five, so a new edge is going to appear there. Three was connected to six, three is connected to four. When I use it, a new four, six is going to appear. So those are a couple of the fill ins. Those are the two fill ins in row four. Then when I go onto row five -- I'll fill in there. So those are the three below the diagonal and above the diagonal, non 0s that come from the bad bordering, I think. I didn't prepare that, so I hope it's right. Yeah

So what's minimum degree? Minimum degree is simply sort of a greedy type algorithm. We have no way to look far, far ahead and see what non 0s are coming way, way down the line, so let's just take it where we are, We have to choose what should be the next pivot. Let's say we take that first pivot. Good.

So what are the degrees of these ? Let me redraw the graph before the fill in. One, two, three, four, five, six. So I've got six nodes. That has a degree 1. It's connected only to one other node. Actually, all these have degree one and this one has degree five. Five edges are going out. So the whole point of minimum degree is, don't pick that one. At each step, create the new graph. The graph expresses a fill in and we're not going to be able to avoid some fill in normally -- but at every point, take the graph as it is and choose as the next node to work on -- that means the next row to use an elimination, the next pivot to choose -- choose one that has smallest degree. Don't choose this one.

So for this very special case, we have five choices for what comes first. We make a choice like that one. We do elimination. In this case, it doesn't fill in anything. Now I'm looking at the rest of the graph. I have four nodes there with degree only 1 and a node there with high degree, so I don't choose that. So I choose 2 and so on. So of course, minimum degree, approximate or perfect or any dumb idea is going to create the right ordering that produces this matrix. I hope you see what minimum degree is. At every stage, you look down the column, just as here. Actually, what would minimum degree give here? I would love to have a complete picture. I think we could at least see at the start.

Let me erase this red-black ordering and prepare to write in a minimum degree order. So what node do I take first? Remember, the graph really does not have these pieces. I might as well erase them now. Shall I just erase -- these are pieces that connect to known values that are on the right hand side. So I think the graph doesn't have those connections, those connections -- it's going to look pretty much as it did before, but I think now we have the graph of actual 16, the graph that we should be looking for. That's the graph of our matrix, the graph of the non 0 structure in the matrix.

So what do you take as the first node? You take the corners, right? The corners are going to win, because they have degree 2. These guys have degree 3. There's a degree 2, so I might take that -- let's see what happens. When I take this one, then I have to see when I use that as the pivot, exactly what I did here -- what's going to fill in? I guess explicitly I know that two new edges fill in and what are they? Maybe they're the same. I guess they are. This is edge number -- on the graph, it's probably easier to see than in the matrix. So I use this guy. By elimination, I'm removing the connections, so it maybe sits there, surviving to remind me which node wasn't numbered one, but these connections have been removed by elimination. That entry became a 0 and that entry became a 0, but there was a fill in here, right?

That's the first step of minimum degree. The second step, as you say, I'm going to do all the corners first, right? Because that has degree 2 and these still have degree 3. No improvement. It's kind of fun to do. Two and that filled in. Number three and that filled in. Number four and that too. What next? There are going to be a bunch of choices now. As far as I know -- I could be wrong -- I don't know whether the -- I don't know what the choice decision -- how the decision is made when there is a choice, when I have one, two, three, four, five, six, seven, eight nodes, all with degree 3. Let me just take this one. So I'll take this as the next one, as number five, I guess. That'll be number five. That will kill all those connections and just leave the node, but it will put in that one. Any others? Maybe only that one. Did I do that right? I mean, it would have put in this one, because those are both connected to this, but that edge is already in there, so nothing new.

Actually, here I've created a node that is also degree 3. So I'm going to have a lot of degree 3s here. After awhile -- I think when I get into the center -- I'm guessing that the degree will go up. I don't think I get away with degree 3 all the way until the finish, I don't think. Anyway, you see the point that there are choices to make among equal node with equal degrees. Then there's this calculation to make -- maybe I should -- this is a moment to just mention, if I can do it, how you might keep -- how you might organize the this pass through -- so this pass through the matrix -- it's really a just a pass through the graph. It's not looking at the actual numbers. It's just choosing the ordering and then the next pass through we follow that ordering with the numbers, finding the multpliers that go into l and the numbers that are left in matrix u.

So how would I -- how would anybody reasonably write down the original list? So I have the node numbered one, two, three, four -- up to sixteen. Those are the node numbers. Now with node one, I'll list all the -- so this corresponds to row one of the matrix, so I'll list all the nodes that are connected to row one, to node one. Node one itself is -- I don't know whether to put that in or not. Maybe I will. Number two is connected to it and number five above it. I'm taking the original row ordering. Then when I look at node two -- this is going to be a long list, but I'll have to have a break there to tell me that these numbers were associated with row one. Now I want to know those associated with row two. So row two had the connection back to one, so this little group is going to be all the things connected to row two. Remember -- sorry, since I've erased all those connections, but you remember them -- two was connected to one, it's connected forward to three and above it to probably six. Maybe that's one, three, and six and then a slash and then whatever node three -- so I need -- I guess I need one more line there to tell me I need a pointer. I need a line, a pointer that tells me, number one, start with the first one -- one, two, three. Number two, start with the fourth one. Number three, start with the seventh one. So that number seven is pointing to the beginning of the sub list, the beginning of the short list that's associated to each separate node.

I guess what I'm saying is that this is a pretty compact description of the graph. This will tell me all the edges in the graph, It tells me what two is connected to and this tells me where that little set lies. Now if I, as I will, change the ordering, then I just reorganize the pointers. I'm going to speak roughly and not the details, just in general, in the large, you could imagine that this is a pretty convenient structure to use for keeping the record of what the notes numering is.

The command nnz is so useful one. That's a MATLAB command that counts the number of non 0s. So n, n and z suggest exactly what that will do. So if you pick an ordering, like approximate minimum degree and you pick a matrix like this one and you run -- I've been very interested in that. You take the matrix, k to d, you take the permutation that comes from approximate minimum degree, that gives you this reordered matrix. We then do elimination, which factors in l times u -- probably l times l transposed, because it's a positive definite. Then I would do nnz of l as a measure of fill in. That would count the fill in in l and with symmetry, of the same in u. If you have a better idea than approximate minimum degree, that's a measure to use as the criteria. I've just a little time to speak finally about this idea of graph separators. So again, I have to get back to a graph. Let me draw a graph under. So now I'm ready for this one. I'll put a little star next to it, to say that it didn't look for a long time is if it could compete with number two, because it'll be a different ordering. You could, in a way -- it's going to be a block minimum degree order. Let me say what it is and as I draw the graph, let me say when it might win. It might win when the size goes way up, 3D. In 2D, minimum degree is -- on sparse matrices, is quite successful. So in two dimensions, until the size really gets big, that's going to be fine. In three dimensions -- I'm still going to draw only two, but in three dimensions or very, very large matrices, I want to think about this graph separator.

So let me -- I'll make it just so I really separate it perfectly -- let me make the number -- so when you look at that graph, you might say, if I -- look at that set of nodes. So a separator is a set of nodes that separates the rest of the nodes into two groups. So I have a group of nodes p, a group of nodes q and nothing, no connections between p and q. P is this part. q is a similar part over there. s, the separator is the part with the connection. So p is connected to q only through s. So now if I think of ordering, in what order should I take the p, the q and the s? Our absolute rule is, everything's connected to s, put it last. So you see that this is the block form of my small example here. I didn't particularly have a p or q, but I certainly wanted to put s last. Maybe I did. So maybe the separator here was -- so what's the separator? One possible separator would be like that, as s, p for these two nodes, q for these three nodes and the rule would be -- I haven't said what I'll do within p or within q or within s, but the rule would be put q last. Of course, that was exactly the right thing to do. We're following the same rule, just [UNINTELLIGIBLE]

So we have the -- so our matrix -- what does our matrix look like when you choose this order? You have whatever p is, the block of all these ps. You have the block of all these qs, which might be connected to each other and r, but not the p. So the main point is, you have these 0 blocks and then everything is connected to s and s is connected to itself. So all you've achieved, but could be considerable, is to take an ordering in which p was -- since p is not connected to q, that produced these large 0 blocks and of course, they'll stay 0 as elimination goes forward. those 0s there will -- I don't have to do any pivoting operations, any row operations to make those 0. They're already 0 and so the only operations will be p on to s, q on to s, s onto itself.

In many problems, it's the s onto itself part -- so what will happen then when I do elimination on s? When I do elimination -- I'll have whatever work is necessary within p, so I'll be factoring -- can I call that p, that part of the matrix? It's an abuse of notation. This is the connections of s to p. This is the connections of s to q, s to itself, q to s, p to s -- probably symmetric. So when I do elimination -- if I don't, if I accept this ordering -- if I accept this ordering and just go for it, elimination will factor p into lu because it'll work up in the corner as it always does -- then there's no connection here, so it'll factor q into its own lu and of course, all that -- all this stuff down here will change, but effectively these will become 0s, because then there's a serious elimination, p onto s, make that 0. q onto s -- q will make that 0. This will be some -- I'll call it sss. It's a much fuller matrix -- block, smaller though, hopefully smaller -- but pretty full. So actually in the graph separator nested dissection world, most of the time is spent on that third block, just working with it, because it's the one that's pretty full.

So how do graph separators work? First of all, you need an algorithm that finds the separator and a lot of computer science attention has gone on to that. Give it a graph, cut it in half. We don't want more nodes in the separator than necessary, because the work is going to end up there. So we want to cut the graph pretty nearly in half by a short separator. If we only did the dissection once, that's what we need, but this idea of nested dissection -- you know what's going to come before I say it. The idea of nested dissection section will be, take this, take the graph -- can I just draw it this way without drawing all these? Cut it by a separator of s 1. That gives a p 1 and a q 1, but now what's coming? You're going to separate p 1. In order to do the work up in that left corner for the p matrix, that's itself a sparse matrix, so now I'll look for a separator and what separator will I choose for the left side of this graph? I'll choose the horizontal separator. This will be the second separator. It'll separate the two bits of p. The third separator will separate the two bits of q. Now I have this part to work on, a fourth separator will -- you'll see its a nester. These separators become nested. There's s 5 there's s 6 and so on.

I think everybody sees in principle what nested dissection should do and the question is, how does it compare with minimum degree? You see, it's just like a different -- we're off on a different route here. You can compute asymptotically for large n, what power of n appears there, what power of n appears in minimum degree -- but I would love to know, does the -- I would be happy if somebody used approximate minimum degree on our model problem and maybe also on some other problem. We could see what power of n -- that's essentially what we're looking for. So it's n to what power for fill in and for the operation count, which the number flops, which controls the cost and the time. So it's what power of n appears there and what power appears in the other method.

There you have it for direct elimination methods. I'll just mention the name of the -- guy's at University of Florida, and he's the one who contributed the code that MathWorks uses and has just announced a book coming on this exact topic, to come later this year from Sci Am. So there will be a book that -- from the person I take to be the leader in this algorithmic development.

Just to remind you -- Friday is my guest, former PhD student who is going to speak about numerical methods for finance -- Black-Scholes equations and other problems to find the value of financial derivatives. See you Friday.