JAMES W. SWAN: So this is going to be our last lecture on linear algebra. The first three lectures covered basics. The next three lectures, we talked about different sorts of transformations of matrices. This final lecture is the last of those three. We're going to talk about in another sort of transformation called the singular value decomposition.

OK, before we jump in, I'd like to do the usual recap business. I think it's always hopeful to recap or look at things from a different perspective. Early on, I told you that the infinite dimensional equivalent of vectors would be something like a function, which is a map, a unique map maybe from a point to x to some value f of x. And there is an equivalent representation of the eigenvalue eigenvector problem in function space. We call these eigenvalues and eigenfunctions.

Here's a classic one where the function is y of x, OK? This is the equivalent of the vector, and equivalent of the transformation or the matrix that's this differential operator this time, the second derivative. So I take the second derivative of this particular function, and the function is stretched. It's multiplied by some fixed value at all points. And it becomes lambda times y.

And that operator has to be closed with some boundary conditions as well. We have to say what the value of y is at the edges of some boundary. So there's a one-to-one correspondence between these things.

What is the eigenfunction here, or what are the eigenfunctions? And what are the eigenvalues associated with this transformation or this operator? Can you work those out really quickly? You learned this at some point, right?

Somebody taught you differential equations and you calculated these things. Take about 90 seconds. Work with the people around you. See if you can come to a conclusion about what the eigenfunction and eigenvalues are.

That's enough time. You can work on this on your own later if you've run out of time. Don't

worry about it. Does somebody want to volunteer a guess for what the eigenfunctions are in this case? What are they? Yeah?

AUDIENCE: [INAUDIBLE]

JAMES W. SWAN: OK, so you chose exponentials. That's an interesting choice. That's one possible choice you can make. OK, so we could say-- this is sort of a classical one that you think about when you first learn differential equation. They say, an equation of this sort has solutions that look like exponentials, and that's true. There's another representation for this, which is as trigonometric functions instead, right?

Either of those is acceptable. [INAUDIBLE] the trigonometric functions, that representation is a little more useful for us here. We know that the boundary conditions tell us that y of 0 is supposed to be 0. That means that the C1 has to be 0, because cosine of 0 is 1. So C1 has 0 in this case. So that fixes one of these coefficients.

And now we're left with a problem, right? Our solutions, our eigenfunctions, cannot be unique. So we don't get to specify C2, right? Any function that's a multiple of this sine should also be an eigenfunction.

So instead the other boundary condition, this y of l equals 0, needs to be used to pin down with the eigenvalue is. So the second equation, y of l equals 0, which implies that the square root of minus lambda has to be equal to 2 pi over l, it has to be all the nodes of the sine where the sine is equal to 0. That's the equivalent of our secular characteristic polynomial that prescribes with the eigenvalues are associated with each of the eigenfunctions.

So now we know what the eigenvalues are. The eigenvalues are the set of numbers minus 2 pi n over l squared. There's an infinite number of eigenvalues. It's an infinite dimensional space that we're in, so it's not a big surprise that it works out that way. And the eigenvectors then are different scalar multiples of sine of the eigenvalues, square root of the eigenvalues, minus x.

There's a one-to-one correspondence between all the linear algebra we've done and linear differential equations or linear partial differential equations. You can think about these things in exactly the same way. I'm sure in 1050, you started to talk about orthogonal functions to represent solutions of differential equations. Or if you haven't, you're going to very soon.

This is a part of the course you get to look at the analytical side of some of these things as opposed to the numerical side. But there's a one-to-one relationship between those things. So if you understand one, you understand the other, and you can come at them from either perspective.

This sort of stuff is useful. Actually, the classical chemical engineering example comes from quantum mechanics where you think about wave functions and different energy levels corresponding to eigenvalues. That's cool. Sometimes, I like to think about a mechanical analog to that, which is the buckling of an elastic column.

So you should do this at home. You should go get a piece of spaghetti and push on the ends of the piece of the spaghetti. And the spaghetti will buckle. Eventually it'll break, but it'll buckle first. It'll bend.

And how does it bend? Well, a balance of linear momentum on this bar would tell you that the deflection in the bar at different points x along the bar multiplied by the pressure has to balance the bending moment in the bar itself. So this e is some elastic constant. I has a moment of inertia.

And D squared y dx squared is something like the curvature of the bar. So it's the bending moments of the bar that balances the pressure that's being exerted on the bar. And sure enough, this bar will buckle when the pressure applied exceeds the first eigenvalue associated with this differential equation.

We just worked that eigenvalue out. We said that that eigenvalue had to be the square root of 2 pi over l squared. And so when the pressure exceeds square root of 2 pi over l squared times the elastic modulus, this column will bend and deform continuously until it eventually breaks, right? It will undergo this linear elastic deformation, then plastic deformation later, and it will break.

The Eiffel Tower, actually, is one of the first structures in the world to utilize this principle, right? It's got very narrow beams in it. The beams are engineered so that their elastic modulus is strong enough that they won't buckle. Gustave Eiffel is one of the first applied physicists, somebody who took the physics of elastic bars and applied them to building structures that weren't big and blocky, but used a minimal amount of material. Cool, right?

OK, so that's recap. Any questions about that? You've seen these things before. You

understood them well before too maybe? Give some thought to this, OK?

We talked about eigendecomposition last time that, associated with the square matrix, was a particular eigenvalue or particular set of eigenvalues, stretches and corresponding eigenvectors directions. These were special solutions to the system of linear equations based on a matrix. It was a square matrix.

And you might ask, well, what happens if the matrix isn't square? What if A is in the space of real matrices that are n by m, where n and m maybe aren't the same? Maybe they are the same, but maybe they're not. And there is an equivalent decomposition. It's called the singular value decomposition. It's like an eigendecomposition for non-square matrices.

So rather than writing our matrix as some w lambda w inverse, we're going to write it as some product U times sigma times V with this dagger. The dagger here is conjugate transpose. Transpose the matrix, and take the complex conjugate of all the elements, OK? I mentioned last time that eigenvalues and eigenvectors could be complex, potentially, right? So whenever we have that case where things can be complex, usually the transposition operation is replaced with the conjugate transpose.

What are these different matrices. Well, let me tell you. U is a complex matrix. It maps from the space N to R N to R N, so it's an n by n square matrix. Sigma is a real valued matrix, and it lives in the space of n by n matrices.

V is a square matrix again, but it has dimensions m by m. Remember, A maps from R M to R N, so that's what the sequence of products says. B maps from m to m. Sigma maps from m to n. U maps from n to n. So this match from m to n as well.

Sigma is like lambda from before. It's a diagonal matrix. It only has diagonal elements. It's just not square, but it only has diagonal elements, all of which will be positive. And then U and V are called the left and right singular vectors. And they have special properties associated with them, which I'll show you right now.

Any questions about how this decomposition is composed or made up? It looks just like the eigendecomposition, but it can be applied to any matrix. Yes?

AUDIENCE: Quick question.

JAMES W. SWAN: Sure.

AUDIENCE: Do all matrices have this thing, or is it like the eigenvalues where some do and some don't.

JAMES W. SWAN: This is a great question. So all matrices are going to have a singular value decomposition. We saw with the eigenvalue decomposition that there could be a case where the eigenvectors are degenerate, and we can't write that full decomposition. All matrices are going to have this decomposition.

So for some properties of this decomposition, U and V are what we call unitary matrices. I talked about these before. Unitary matrices are ones for whom, if they're real valued, their transpose is also their inverse.

If they're complex valued, and they're conjugate transpose is the equivalent of their inverse. So U times U conjugate transpose will be identity. V times V conjugate transpose will be identity.

Unitary matrices also have the property that they impart no stretch to a matrix-- or to vectors. So their maps don't stretch. They're kind of like rotational matrices, right? They change directions, but they don't stretch things out.

If I were to take A conjugate transpose and multiply it by A, that would be the same as taking U sigma V conjugate transpose, and multiplying it by U sigma V. If I use the properties of matrix multiplications and complex conjugate transposes, and work out what this expression is, I'll find out that it's equivalent to V sigma conjugate transpose sigma V conjugate transpose.

Well this has exactly the same form as an eigendecomposition. An eigendecomposition of A times A instead of an eigendecomposition of A. So V is the set of eigenvectors of A conjugate transpose A, and sigma squared are the eigenvalues of A conjugate transpose times A.

And if I reverse the order of this multiplication-- so I do A times A conjugate transpose-- and work it out, that would be U sigma sigma U. And so U are the eigenvectors of A A conjugate transpose, and sigma squared are still the eigenvalues of A A conjugate transpose. So what are these things U and V? They relate to the eigenvectors of the product of A with itself, this particular product of A with itself, or this particular product of A with itself.

Sigma are the singular values. And all matrices possess this sort of a decomposition. They all have a set of singular values and singular vectors. These sigmas are called the singular values of the A. They have a particular name. I'm going to show you how you can use this

decomposition to do something you already know how to do, but how to do it formally.

What are some properties of the singular value decomposition? So if we take a matrix A and we compute it's singular value decomposition, this is how you do it in Matlab. We'll find out, for this matrix, U is identity. Sigma is identity with an extra column pasted on it. And B is also identity. I mean, this is the simplest possible four by three matrix I can write down.

You don't have to know how to compute the singular value decomposition, you just need to know that it can be computed in this way. You might be able to guess how to compute it based on what we did with eigenvalues earlier and eigenvectors. It'll turn out some of the columns of sigma will be non-zero right? There are three non-zero columns of sigma. And the columns of V, they correspond to those columns of sigma, spanned the null space of the matrix A.

So the first three columns here are non-zero, the first three columns of V. I'm sorry, the first three columns here are non-zero. The last column is 0.

The columns of sigma which are 0 correspond to a particular column in V, this last column here, which lives in the null space of A. So you can see, if I take A and I multiply it by any vector that's proportional to 0, 0, 0, 1, I'll get back 0. So the null space of A is spanned by all these vectors corresponding to the 0 columns of sigma.

Some of the columns of sigma are non-zero. These first three columns. And the rows of U corresponding to those three columns span the range of A. So if I do the singular value decomposition of a matrix, and I look at U, V, and sigma and what they're composed of-- where sigma is 0 and non-zero, and the corresponding columns or rows of U and V-- then I can figure out what vectors span the range and null space of the matrix A.

Here's another example. So here I have A. Now instead of being three rows by four columns, it's four rows by three columns. And here's the singular value decomposition that comes out of Matlab.

There are no vectors that live in the null space of A, and there are no 0 columns in sigma. There's no corresponding columns in V. There are no vectors in the null space of A.

The range of A is spanned by the rows corresponding to the non-zero-- the rows of U corresponding to the non-zero columns of sigma. So it's these three columns in the first three rows. And these first three rows, clearly they span-- they describe the same range as the three columns in A.

So the singular value decomposition gives us direct access to the null space and the range of a matrix. That's handy. And it can be used in various ways. So here's one example where it can be used.

Here I have a fingerprint. It's a bitmap. It's a square bit of data, like a matrix, and each of the elements of the matrix takes on a value describing how dark or light that pixel. Let's say it's grayscale, and it's value's between 0 and 255. That's pretty typical.

So I have this matrix, and each element to the matrix corresponds to a pixel. And I do a singular value decomposition. Some of the singular values, the values of sigma, are bigger than others. They're all positive, but some are bigger than others. The ones that are biggest in magnitude carry the most information content about the matrix.

So we can do data compression by neglecting singular values that are smaller than some threshold, and also neglecting the corresponding singular vectors. And that's what I've done here. So here's the original bitmap of the fingerprint. I did the singular value decomposition, and then I retained only the 50 biggest singular values and I left all the other singular values out.

This bitmap was something like, I don't know, 300 pixels by 300 pixels, so there's like 300 singular values, but I got rid of 5/6 of the information content. I dropped 5/6 of the singular vectors, and then I reconstructed the matrix from the singular values and those singular vectors, and you get a faithful representation of the original fingerprint.

So the singular value decomposition says something about the information content in the transformation that is the matrix, right? There are some transformations that are of lower power or importance than others. And the magnitude of these singular values tell you what they are. Does that makes sense?

How else can it be used? Well, one way it can be used is finding the least square solution to the equation Ax equals b, where A is no longer a square matrix, OK? You've done this in other contexts before where the equations are overspecified. We have more equations than unknowns, like data fitting. You form the normal equations, you multiply both sides of Ax equals b by A transpose, and then invert A transpose A.

You might not be too surprised, then, to think that singular value decomposition could be

useful here too. Since we already saw the data in a singular value decomposition corresponds to eigenvectors and eigenvalues of this A transpose A, right? But there's a way to use this sort of decomposition formally to solve problems that are both overspecified and underspecified.

Least squares means find the vector of solutions x that minimizes this function phi. Phi is the length of the vector given by the difference between Ax and b. It's one measure of how far an error our solution x is. So let's define the value x which is least in error. This is one definition of least squares.

And I know the singular value decomposition of A. So A is U sigma times V. So I have U sigma V times x. I can factor out U, and I've got a factor of U transpose, or U conjugate transpose multiplying by b. So Ax minus b is the same as U times the quantity sigma V conjugate transpose x minus U conjugate transpose b.

We want to know the x that minimizes this phi. It's an optimization problem. We'll talk in great detail about these sorts of problems later.

This one is so easy to do, we can just work it out in a couple lines of text. We'll define a new set of unknowns, y, which is V transpose times x, and a new right-hand side for a system of equations p, which is U transpose times b. And then we can rewrite our function phi that we're trying to minimize.

So phi then becomes U sigma y minus p. U is a unitary vector. It imparts no stretch in the two norms, so this sigma y minus p doesn't get elongated by multiplication with U.

So it's length, the length of this, is the same as the length of sigma y minus p. You can prove this. It's not very difficult to show at all. You use the definition of the two norm to prove it.

So phi is minimized by y's, which makes this norm smallest, make it closest to 0. Let r be the number of non-zero singular values, the number of those sigmas which are not equal to 0. That's also the rank of A.

Then I can rewrite phi as the sum from i equals 1 to r of sigma i i time y i minus p i squared. That's parts of this length, this Euclidean length, for which sigma is non-zero. Plus the sum from r plus 1 to n, the sum over the rest of the values of p, for which the corresponding sigmas are 0.

I want to minimize phi, and the only thing that I can change to minimize it is what? What am I

free to pick in this equation in order to make phi as small as possible? Yeah?

AUDIENCE: y.

JAMES W. SWAN: y, so I need to choose the y's that make this phi as small as possible. What value should I choose for the y's? What do you think?

AUDIENCE: [INAUDIBLE]

JAMES W. SWAN: Perfect, right? Choose y equals p i over sigma i i. Right, y i is p i over sigma i i. Then all of these terms is 0. I can't make this sum any smaller than that. That fixes the value of y i up to r.

I can't do anything about this left over bit here. There's no choice of y that's going to make this part and the smaller. It's just left over. It's some remainder that we can't make any smaller or minimize an smaller. There isn't an exact solution to this problem, in many cases.

But one way this could be 0 is if r is equal to n. Then there are left over unspecified terms, and then this y i equals p i over sigma i is the exact solution to the problem. So this is what you told me.

Choose y i is p i over sigma i i for i bigger than 1 and smaller than r. There are going to be values of y i that go between r plus 1 and m, because A was a vector that mapped from m to n, right? So I have extra values of y that could be specified potentially.

If that's true, if r plus 1 is smaller than m, then there's some components of y that I don't get to-- I can't specify, right? My system of equations is somehow underdetermined. I need some external information to show me what values to pick for those y i. I don't know. I can't use them.

Sometimes people just set y i equal to 0. That's sort of silly, but that's what's done. It's called the minimum norm least square solution. y has minimum length, when you set all these other components to 0. But the truth is, we can't specify those components, right? We need some external information in order to specify them.

Once we know y, we can find x going back to our definition of what y is. So I multiply this equation by V on both sides, and I'll get V y equals x. So I can find my least square solution to the problem from the singular value decomposition. So I can find the least square solution to both overdetermined and underdetermined problems using singular value decomposition.

It inherits all the properties you know of solving the normal equations, multiplying by A transpose the entire equation, and solving for a least square solution that way. But that's only good for overdetermined systems of equations. This can work for underdetermined equations as well. And maybe we do have extraneous information that lets us specify these other components somehow. Maybe we do a separate optimization that chooses from all possible solutions where these y i's are free, and picks the best one subject to some other constraint.

Does it makes sense? OK, that's the last decomposition we're going to talk about. It's as expensive to compute the singular value decomposition as it is to solve a system of equations. You might have guessed that it's got an order N cubed flavor to it.

It's kind of inescapable that we run up against those computational difficulties, order N cubed computational complexity. And there are many problems of practical interest, particularly solutions of PDEs, for which that's not going to cut it. Where you couldn't solve the problem with that sort of scaling in time. You couldn't compute the Gaussian elimination, or the singular value decomposition, or an eigenvalue decomposition. It won't work.

And in those cases, we appeal to not exact solution methods, but approximate solution methods. So instead of trying to get an exact solution, we'll try to formulate one that's good enough. We already know the computer introduces numerical error anyways. Maybe we don't need machine precision in our solution or something close to machine precision in our solution. Maybe we're solving engineering problem, and we're willing to accept relative errors on the order of 10 to the minus 3 or 10 to the minus 5, some specified tolerance that we apply to the problem.

And in those circumstances, we use iterative methods to solve systems of equations instead of exact methods, elimination methods, or metrics decomposition methods. These algorithms are all based on iterative refinement of an initial guess. So if we have some system of equations we're trying to solve, Ax equals b, we'll formulate some linear map, right? xi plus 1 will be some matrix C times x i plus some little vector c where x i is my last best guess for the solution to this problem, and x i plus 1 is my next best guess for the solution to this problem. And I'm hoping, as I apply this map more and more times, I'm creeping closer to the exact solution to the original system of equations.

The map will converge when x i plus 1 approaches x i, when the map isn't making any changes to the vector anymore. And the converged value will be a solution when x i-- which is

equal to i minus c inverse times c, if I replace x i was 1 with x i appear, so I say that my map has converged-- when this value is equivalent to A inverse times B, when it's a solution to the original problem, right? So my map may converge. It may not converge to a solution of the problem I like, but if it satisfies this condition, then has converged to be a solution of the problem that I like as well.

And so it's all about using this C here and this little c here so that this map converges to solution of the problem I'm after. And there are lots of schemes for doing this. Some of them are kind of ad hoc. I'm going to show you one right now. And then when we do optimization, we'll talk about a more formal way of doing this for which you can guarantee very rapid convergence to a solution.

So here's a system of equations I'd like to solve. It's not a very big one. It doesn't really make sense to solve this one iteratively, but it's a nice illustration. One way to go about formulating this map is to split this matrix into two parts.

So I'll split it into a diagonal part and an off diagonal part. So I haven't changed the problem at all by doing that. And then I'm going to rename this x x i plus 1, and I'm going to rename this x x i. And then move this matrix vector product to the other side of the equation.

And here's my map. Of course, this matrix multiplied doesn't make any-- it's not useful to write it out explicitly. This is just identity. So I can drop this entirely. This is just x i plus one.

So here's my map. Take an initial guess, multiply it by this matrix, add the vector 1, 0, and repeat over and over and over again. Hopefully-- we don't really know-- but hopefully, it's going to converge to a solution of the original linear equations.

I didn't make up that method. That's a method called Jacobi Iteration. And the strategy is to split the matrix A into two parts-- a sum of its diagonal elements, and it's off diagonal elements-- and rewrite the original equations as an iterative map.

So D times x i plus 1 is equal to minus r times x i plus b. Or x i plus 1 is D inverse times minus r x i plus b. If the equations converge, then D plus r times x i has to be equal to b, we will have found a solution. If it converges, right? If these iterations approach a steady value. If they don't change from iteration to iteration. Is

The nice thing about the Jacobi method is it turns the hard problem, the order N cubed problem of computing A inverse B, into a succession of easy problems, D inverse times some

vector C. How many calculations does it take to compute that D inverse? N, that's right, order N.

It's just a diagonal matrix. I invert each of its diagonal elements, and I'm done. So I went from order N cubed, which was going to be hard, into a succession of order N problems. So as long as it doesn't take me order N squared iterations to get to the solution that I want, I'm going to be OK. This is going to be a viable way to solve this problem faster than finding the exact solution.

How do you know that it converges? That's the question. Is this thing actually going to converge or not, or are these iterations just going to run on and on forever? Well, one way to check whether it will converge or not is to go back up to this equation here, and substitute b equals Ax, where x is the exact solution to the problem.

And you can transform, then, this equation into one that looks like x i plus 1 minus x equal to minus D inverse times r x i minus x. And if I take the norm of both sides and I apply our normal equality-- where the norm of a matrix vector product is smaller than the product of the norms of the matrices of the vectors-- then I can get a ratio like this. That the absolute error in iteration I plus 1 divided by the absolute error in iteration i is smaller than the norm of this matrix.

So if I'm converging, then what I expect is this ratio should be smaller than 1. The error in my next approximation should be smaller than the error in my current approximation. That makes sense? So that means that I would hope that the norm of this matrix is also smaller than 1.

If it is, then I'm going to be guaranteed to converge. So for a particular coefficient matrix, for a system of linear equations I'm trying to solve, I may be able to find-- I may find that this is true. And then I can apply this method, and I'll converge to a solution.

We call this sort of convergence linear. Whatever this number is, it tells me the fraction by which the error is reduced from iteration to iteration. So suppose this is 1/10. Then the absolute error is going to be reduced by a factor of 10 in each iteration. It's not going to be 1/10 usually. It's going to be something that's a little bit bigger than that typically, but that's the idea.

You can show-- I would encourage you to try to work this out on your own-- but you can show that the infinity norm of this product-- infinity norm of this product is equal to this. And if I ask

that the infinity norm of this product be smaller than 1, that's guaranteed when the diagonal values of the matrix and absolute value are bigger than the sum of the off diagonal values in a particular row or a particular column. And that kind of matrix we call diagonally dominant. The diagonal values are bigger than the sum and absolute value of the off diagonal pieces.

So diagonally dominant matrices, which come up quite often, can be-- those linear equations based on those matrices can be solved reasonable efficiency using the Jacobi method. There are better methods to choose. I'll show you one in a second. But you can guarantee that this is going to converge to a solution, and that the solution will be the right solution to the linear equations you were trying to solve.

So if the goal is just to turn hard problems into easier to solve problems, then there are other natural ways to want to split a matrix. So maybe you want to split into A lower triangular part which contains the diagonal elements of A, and an upper triangular part which has no diagonal elements of A. We just split this thing apart. And then we could rewrite our system of equations is an iterative map like this, L times x i plus 1 is minus U times x i plus b.

All I have to do is invert l to find my next iteration. And how expensive computationally is it to solve a system of equations which is triangular? This is a process we call back substitution. Its order--

AUDIENCE: N squared.

JAMES W. SWAN: --N squared. So we still beat N cubed. One would hope that it doesn't require too many iterations to do this. But in principle, we can do this order N squared operations many times. And it'll turn out that this sort of a map converges to the solution that we're after.

It converges when matrices are either diagonally dominant as before, or they're symmetric and they're positive definite. Positive definite means all the eigenvalues of the matrix are bigger than 0. So try the iterative method solving some equations and see how we convert. Yes?

AUDIENCE: How do you justify ignoring the diagonal elements in that method?

JAMES W. SWAN: So the question was, how do you justify ignoring the diagonal elements in this method. Maybe I was going too fast or I misspoke. So I'm going to split A into a lower triangular matrix that has all the diagonal elements, and U is the upper parts with none of those diagonal elements on it.

Does that make sense?

AUDIENCE: Yeah.

JAMES W. SWAN: Thank you for asking that question. I hope that's clear. l holds onto the diagonal pieces and U takes those away. So let's try it.

On a matrix like this, the exact solution to this system of equations is 3/4, 1/2, and 1/4. All right, we'll try Jacobi, we'll have to give it some initial guess for the solution, right? We're talking about places where you can derive those initial guesses from later on in the course, but we have to start the iterative process with some guess at the solutions.

So here's an initial guess. We'll apply this map. Here's Gauss-Seidel with the same initial guess, and we'll apply this map. They're both linearly convergent, so the relative error will go down by a fixed factor after each iteration.

Iteration one, the relative error in Jacobi will be 38%. In Gauss-Seidel, it'll be 40%. If we apply this all the way down to 10 iterations, the relative error Jacobi will be 1.7%, and the relative error in Gauss-Seidel 0.08%.

And we can go on and on with these iterations if we want until we get sufficiently converged, we get to a point where the relative error is small enough that we're happy to accept this answer as a solution to our system of equations. So we traded the burden of doing all these calculations to do elimination for a faster, less computationally complex methodology. But the trade off was we don't get an exact solution anymore. We're going to have finite precision in the result, and we have to specify the tolerance that we want to converge to.

We're going to see now-- this is the hook into the next part of that class-- we're going to talk about solutions of nonlinear equations next for which there are almost no non-linear equations that we can solve exactly. They all have to be solved using these iterative methods. You can use these iterative methods for linear equations. It's very common to do it this way.

In my group, we solve lots of systems of linear equations associated with hydrodynamic problems. These come up when you're talking about, say, low Reynolds number flows, which are linear sorts of fluid flow problems. They're big. It's really hard to do Gaussian elimination, so you apply different iterative methods.

You can do Gauss-Seidel. You can do Jacobi. We'll learn about more advanced ones like

PCG, which you're applying on your homework now, and you should be seeing that it converges relatively quickly in cases where exact elimination doesn't work. We'll learn, actually, how to do that method. That's one that we apply in my own group. It's pretty common to use out there. Yes?

AUDIENCE: One question, is that that Gauss, [INAUDIBLE]

JAMES W. SWAN: Order N squared.

AUDIENCE: Yeah, that's what I meant. So now we've got an [INAUDIBLE]. So we basically have [INAUDIBLE] iterations, right?

JAMES W. SWAN: This is a wonderful question. So this is a pathological problem in the sense that it requires a lot of calculations to get an iterative solution here. We haven't gotten to an end that's big enough that the computational complexities crossover. So for small Ns, probably the factor in front of N-- whatever number that is-- and maybe even the smaller factors, order N squared factors on that order N cubed, play a big role in how long it takes to actually complete this thing. But modern problems are so big that we almost always are running out to Ns that are large enough that we see a crossover.

You'll see this in your homework this week. You won't see this crossover at N equals 3. You're going to see it out at N equals 500 or 1,200, big problems. Then we're going to encounter this crossover. That's a wonderful question. So first small system sizes, iterative methods maybe don't buy you much.

I suppose it depends on the application though, right? If you're doing something that involves solving problems on embedded hardware, in some sort of sensor or control valve, there may be very limited memory or computational capacity available to you. And you may actually apply an iterative method like this to a problem that that controller needs to solve, for example. It just may not have the capability of storing and inverting what we would consider, today, a relatively small matrix because the hardware doesn't have that sort of capability.

So there could be cases where you might choose something that's slower but feasible, versus something that's faster and exact, because there are other constraints. They do exist, but modern computers are pretty efficient. Your cell phone is faster than the fastest computers in the world 20 years ago. We're doing OK. So we've got to get out to big system sizes, big problem sizes, before this starts to pay off. But it does for many practical problems.

OK I'll close with this, because this is the hook into solving nonlinear equations. So I showed you these two iterative methods, and they kind of had stringent requirements for when they were actually going to converge, right? I had to have a diagonally dominant system of equations for Jacobi to converge. I had to have diagonal dominance or symmetric positive definite matrices. These things exist and they come up in lots of physical problems, but I had to have it in order for Gauss-Seidel to converge.

What if I have a system of equations that doesn't work that way? Or what if I have an iterative map that I like for some reason, but it doesn't appear to converge? Maybe it converges under some circumstances, but not others. Well, there's a way to modify these iterative maps, called successive over-relaxation, which can help promote convergence.

So suppose we have an iterative map like this, x i plus 1 is some function of the previous iteration value. Doesn't matter what it is. It could be linear, could be non-linear. We don't actually care.

The sought after solution is found when x i plus 1 is equal to x i. So this map is one the convergence to the exact solution of the problem that we want. We've somehow guaranteed that that's the case, but it has to converge.

One way to modify that map is to say x i plus 1 is 1 minus some scalar value omega times x i plus omega times f. You can confirm that if you substitute x i plus 1 equals x i into this equation, you'll come up with the same fixed points of this iterative map x i is equal to f of x i. So you haven't changed what value will converge here, but you've affected the rate at which it converges.

Here you're saying x i plus 1 is some fraction of my previous solution plus some fraction of this f. And I get to control how big those different fractions. So if things aren't converging well for a map like this, then I could try successive over-relaxation, and I could adjust this relaxation parameter to be some fraction, some number between 0 and 1, until I start to observe convergence. And there are some rules one can use to try to promote convergence with this kind of successive over-relaxation.

This is a very generic technique that one can apply. If you have any iterative map you're trying to apply, it should go to the solution you want but it doesn't converge for some reason, then you can use this relaxation technique to promote convergence to the solution. You may slow

the convergence way down.

It may be very slow to converge, but it will converge. And after all, an answer is better than no answer, no matter how long it takes to get it. So sometimes you've got to get these things by hook or by crook.

So for example, you can apply this to Jacobi. This was the original Jacobi map. And we just take that. We add 1 minus omega times x i plus omega times this factor over here. And now we can choose omega so that this solution converges. We always make omega small enough so that the diagonal values of our matrix appear big enough that the matrix looks like it's diagonally dominated.

You could go back to that same convergence analysis that I showed you before and try to apply it to this over-relaxation form of Jacobi and see that, while there's always going to be some value of omega that's small enough, that this thing will converge. It will look effectively diagonally dominant, because omega inverse times D will be big enough, or omega times D inverse will be small enough. Does that make sense? You can apply the same sort of damping method to Gauss-Seidel as well. It's very common to do this.

The relaxation parameter acts like an effective increase in the eigenvalues of the matrix. So you can think about L. That's a lower triangular matrix. It's diagonal values are its eigenvalues. The diagonal values of L inverse-- well, 1 over those diagonal values are the eigenvalues of L inverse.

And so if we make omega very small, then we make the eigenvalues of L inverse very small, or the eigenvalues or L very big. And again, the matrix starts to look diagonally dominated. And you can promote convergence in this way.

So even though this may be slow, you can use it to guarantee convergence of some iterative procedures, not just for linear equations, but for non-linear equations as well. And we'll see, there are good ways of choosing omega for certain classes of non-linear equations. We'll apply Newton-Raphson method, and then will damp it using exactly the sort of procedure. And I'll show you how you can choose a nearly optimal value for omega to promote convergence to the solution.

Any questions? No, let me address one more thing before you go. We've scheduled times for the quizzes. They are going to be in the evenings on the dates that are specified on the

syllabus.

We wanted to do them during the daytime. It was really difficult to schedule a room that was big enough for this class, so they have to be from 7:00 to 9:00 in the gymnasium. I apologize for that. We spent several days looking around trying to find a place where we could put everybody so you would all get the same experience in the quiz.

I know that the November quiz comes back to back with the thermodynamics exam as well. That's frustrating. Thermodynamics is the next day. That week is tricky. That's AICHE, so most of the faculty have to travel. We won't be able to teach, but you won't have classes one of those days so you have extra time to study. And Columbus Day also falls in that week, so there's no way to put three exams in four days without having them come right back to back.

Believe me, we thought about this and tried to get things scheduled as efficiently as we could for you, but sometimes there are constraints that are outside of our control. But the quiz times are set. There's going to be done in October and one in November. They'll be in the evening, and they'll be in the gymnasium. I'll give you directions to it before the exam, just say you know exactly where to go, OK? Thank you, guys.