OK, we're moving on. No more linear algebra. We're going to try solving some more difficult problems-- of course, all those problems will just be turned into linear algebra as we move on, so your expertise now with different techniques from linear algebra is going to come in handy.

The next section of this course, we're talking about systems of nonlinear equations, and we'll transition into problems in optimization-- which, turns out, look a lot like systems of nonlinear equations, as well. And we'll try to leverage what we learn in the next few lectures to solve different optimization problems.

Before going on, I just want to recap. All right, last time we talked about singular value decomposition-- which is like an eigenvalue decomposition for any matrix. And associated with that matrix are singular vectors, left and right singular vectors. And the singular values of that matrix.

Your TA, Kristen, reminded me that you can actually define a condition number for any matrix, as well. The condition we gave originally was associated with solving square systems of equations that actually have a solution. But there is a condition number associated with any matrix, and it's defined in terms of its singular values. So if you go back and look at the definition of the two norm, you try to think about the condition number associated with the two norm, you'll see that the conditions of any matrix maybe can be defined as the square root of the ratio of the biggest and the smallest singular values of that matrix.

So there's a condition number associated with any matrix. The condition number as an entity makes most sense when we're thinking about how we amplify errors-- numerical errors, solving systems of equation zones. It's most easily applied to square systems that actually have unique solutions, but you can apply it to any system of equations you want. And the singular value decomposition is one way to tap into that.

OK, the last thing we did discussing linear algebra was to talk about iterative solutions, the systems of linear equations. And that's actually our hook into solutions to systems of nonlinear equations. It's going to turn out that exact solutions are hard to come by for anything but linear systems of equations.

And so we're always going to have these iterative algorithms, where we refine initial guesses for solutions until we converge to something that's a solution to the problem that we wanted before. And one question you should ask, when you do these sorts of iterations, is when do I stop? I don't know the exact solution to this problem. I can't say I'm close enough-- what does close enough even mean? So how do I decide to stop?

Do you have any ideas or suggestions for how you might do that? You've done some of this already in a homework assignment. But what do you think? How do you decide to stop? Yeah?

AUDIENCE: [INAUDIBLE].

PROFESSOR: OK, so this is one suggestion-- look at my current iteration, and my next iteration, ask how far apart are these two numbers? If they're sufficiently far apart, seems like I've got some more steps to make before I converge to my solution. And if they're sufficiently close together, the steps I'm taking are small enough that I might be happy accepting this solution as a good approximation.

That's called the step norm criteria-- I'll give you a formalization of that later-- it's called the step norm criteria. How big are the steps that I'm taking? And are they sufficiently small that I don't care about any future steps? Another suggestion?

AUDIENCE: I've got a question.

PROFESSOR: Yeah?

AUDIENCE: [INAUDIBLE] absolute [INAUDIBLE] when we did that for homework, I tried to do it [INAUDIBLE], and [INAUDIBLE].

PROFESSOR: Yes. I will show you a definition of the step norm criteria that integrates both relative and absolute error into the definition. And we'll see why, OK?

One problem may be-- what if the solution I'm trying to converge to is 0? How do you define the relative error with respect to the number 0? There isn't one-- there is only absolute error when you're trying to converge to 0. So you may want to have some measure of both absolute and relative step size in order to determine whether you're converged.

Is that the only way to do it, though? Have any ideas, alternative proposals for deciding convergence?

AUDIENCE: [INAUDIBLE] the residual.

PROFESSOR: The residual. OK, what's the residual?

AUDIENCE: [INAUDIBLE]

PROFESSOR: Good, OK, so we asked, how good a solution-- we can ask how good a solution is this value that we've converged to by putting it back into the original system of equations, and asking, how far out of balance are we? I take my best guess for solution x, and multiply it by and A, and I subtract b-- we call that the residual.

And is the residual sufficiently converged, or not? If it's small enough in magnitude, then we would say, OK, maybe we're sufficiently close to the solution. If it's too big, then we say, OK, let's iterate some more until we get there. That is called the function norm criterion.

We're going to talk about these in detail, as applied to systems of nonlinear equations. But these same criteria apply to all iterative processes. Neither is preferred over the other. You don't know the exact solution-- you have no way of measuring how close or far away you are from the exact solution.

So usually you use as many tools as possible to try to judge how good your approximation is. But you don't know for certain.

What do you do, when that's the case? We haven't really talked about this in this class. You have a problem, you program it into a computer, you get a solution. Is at the end of the story? We just stop? You get a number back out, and that's the answer?

How do you know you're right? How do you know? We talked about numerical error-- every calculation has a numerical error-- how do you know you got the right answer?

What do you think?

AUDIENCE: [INAUDIBLE]

PROFESSOR: OK, yeah, this is true-- so you plug your solution back into the equation, you ask, how good a job does it do satisfying that? But maybe this equation is relatively insensitive to the solution you provide. Maybe many solutions nearby look like they also satisfy the equation, but those solutions are sufficiently far apart. So how do you--

AUDIENCE: [INAUDIBLE]

PROFESSOR: OK, so that sort of physical reasoning is a good one. In your transfer class, you'll talk about doing asymptotic expansions, or asymptotic solutions to problems. Solve this complicated problem in a limit where it has some analytical solution, and figure how the solution scales, with respect to different parameters. So you can have an analytical solution that you compare against your numerical solution in certain limits.

You have experiments that you've done. Experiment-- that's the reality. The computer is a fiction that's trying to model reality, so you can compare your solution to experiments.

You could also solve the problem a bunch of different ways, and see if all these answers converge in the same place. So we're going to talk about solving nonlinear equations-- we're going to need different initial guesses for our iterative methods. We might try several different initial guesses, and see what solutions we come up with. Maybe we converge all to the same solution, or maybe this problem has some weird sensitivity in it, and we get lots of different solutions that aren't coordinated with each other.

One of the duties of someone who's using numerical methods to solve problems is to try to validate their result-- by solving it multiple times or multiple ways, or comparing against experiment, or against known analytical results, and certain limits where the answer should be exact.

But you can't just accept what the computer tells you-- you have to validate it against some sort of external solution that you can compare with. Sometimes it's hard to come up with that solution, but it's immensely important. We know every calculation can be an error. And as we go on to more complicated problems, it's even more important to validate things.

So, systems of nonlinear equations-- so these are problems of a type f of x equals 0, where x is some vector of unknowns, and has dimension N. And f is a function that takes as input vectors of dimension N, and gives an output a vector of dimension N. It's a map from R N to R N. But it's no longer necessarily a linear map-- it can be some nonlinear function of all the elements of this x.

And the solution to this equation, the particular solution of this equation, x-- for which f of x equals 0-- are called the roots of this vector-valued function. The linear equations, then, are just represented in this form as A x minus b, A x minus b equals 0-- it's the same as the linear equations we were solving before. So we're searching for the roots of these functions.

Common chemical engineering examples include equations of state, often nonlinear, in terms of the variables we're interested in. Energy balances have lots of non-linearities introduced into them. Mass balances with nonlinear reactions. Or reactions that are non-isothermal, so their kinetics are sensitive to temperature, and temperatures are variable, we want to know.

These sorts of nonlinear equations crop up all over the place, and you want to be able to solve them reliably. Here's a simple, simple example. The Van der Waals equation of state-- here I've written it in terms of reduced pressure, temperature, and molar volume. Nonetheless, this is the Van der Waals equation of state.

And somebody told you once that if I plot pressure versus molar volume for different temperatures, I may see that, at a given pressure, there could be just one root-- one possible molar volume that satisfies the equation of state. Or there can be one, two, three potential roots.

It turns out we don't know, with nonlinear equations, how many possible solutions there are. We knew, for linear equations, I either had zero, one, or an infinite number of solutions. But with nonlinear equations, in general, there's no way to predict them.

This problem can be transformed into a polynomial, and polynomials are one of the few nonlinear equations where we know how to bound the possible number of solutions.

So we can transform this nonlinear equation to the form I showed you before-- we just move 8/3 T to the other side of this equation. We want to find the roots of this equation. So possibly, given pressure and temperature-- pressure and temperature-- find all the molar volumes that satisfy the equation of state.

This is actually overly simplified for a particular physical problem, of looking at vapor liquid coexistence of the Van der Waals fluid. You can't specify pressure and temperature independently-- the saturation pressure, the coexistence pressure depends on the temperature as the Gibbs phase rule.

So actually, phase equilibria is made up of three parts-- there's thermal equilibrium. I'm going to add two phases, a gas and a liquid, and they have to have the same temperature, otherwise they're not in equilibrium. There's got to be mechanical equilibrium of two phases, the gas and the liquid. They better have the same pressure, otherwise one is going to be pushing harder on the other one. They'll be in motion-- that's not in equilibrium.

They've got to have the same chemical potential-- they have to be in chemical equilibrium. So there can't be any net mass flux from one phase to another, otherwise, one phase is going to grow and the other is going to shrink. They're not in equilibrium with each other.

So actually, the problem of determining vapor liquid coexistence, in this Van der Waals fluid, involves satisfying a number of different equations, some of which are nonlinear, and are constrained by the equation of state.

Given the temperature, there are three unknowns-- the pressure, and the more volumes of the gas and the liquid. And there are three nonlinear equations we have to solve. Two of those are the equation of state, in the gas and the liquid-- I'll show them to you in a second-- and the other is this Maxwell equal area construction, which essentially says that the chemical potential in the two phases is equal. This is one way of representing that.

So we have to solve this system of nonlinear equations for the saturation pressure, the molar volume of the gas or vapor, and the molar volume of the liquid. And these are those equations. Here's the equation of state and the gas, here's the equation of state in the liquid, and here's the Maxwell equal area construction at defined values of P sat, V G and V L that satisfy all three equations. There's not going to be an analytical way to do this-- it has to be done numerically.

Here's a simplification that I can make, though. So I can take that equal area construction, and I can solve for P sat in terms of V G and V L. And so that reduces the dimensionality of these equations from three to two. And when it's two dimensional, I can plot these things, so that's helpful.

So let's plot f 1-- the equation of state in the gas is a function of V G and V L. Where that's equal to 0, that's this black curve here. Let's plot f 2-- the equation of state in the liquid as a function of V G and V L, where that's equal to 0, that's this blue curve here. And the solutions are where these curves intersect. So we're seeking out the specific points graphically where these curves intersect.

First, this solution, and that solution aren't the solutions we're interested in at all. That would say that the molar volume of the gas and the liquid is the same-- that's not really phase separation. We want these heterogeneous solutions out here.

So we need some methodology that can reliably take us to those solutions. We'll see that that methodology-- the most reliable methodology, and one of the ones that converges fastest to the solutions-- is called the Newton-Raphson method. But even before we do that, let's talk more about the structure of systems of nonlinear equations, and what sort of solutions we can expect.

Does this example makes sense to everyone? Have you thought about this before, maybe? Yeah.

So given a function, which is a map from R N to R N, find the special solution, x star, such that f of x star equals 0. That's our task. And there could be no solutions. There can be one to an infinite number of locally unique solutions, and there can be an infinite number of solutions.

A solution is to be locally unique if I can wrap that solution by some ball of points, in which there are no other solutions. That ball can be very, very small, but as long as I can wrap that solution in some ball of points, which are not solutions, we term that locally unique.

So consider the simple function, one which depends on two variables-- x1 and x2, and f 2, which depends on x1 and x2, equals 0. And I'm going to plot, in the x1, x2 plane, were f 1 and f 2 are equal to 0, so that these curves here. And here, we have a locally unique solution-- we see the curves cross at exactly one point. Here, you can see these two curves are tangent with each other.

They could be coincident with each other, over some finite distance. In which case there's a lot of solutions that live on some locally tangent area. Or they could just touch in one point. So they may be tangent, and the solutions there are not locally unique, or they may touch at one point, and the solutions are-- there's one solution, and it's locally unique there.

The reason why we talk about locally unique solutions is it's going to be hard for a numerical method to find anything that's not locally unique in a reliable way. Locally unique solutions, numerical methods can find very reliably. But if they're not locally unique? My iterative method could converge to any one of these solutions that lives on this line, any of these tangent points. And I'm going to have a hard time predicting which one it's going to go to. That's a problem if you're trying to solve something reliably over and over again-- if I converge to one of these solutions, or another solution, or another solution, the data that comes out of that process isn't going to be easy to interpret.

There's something called the inverse function theorem, which says if f of x star is equal to 0, and the determinant of this matrix, J, which we call the Jacobian, evaluated at x star is not equal to 0, then x star necessarily is a locally unique solution. So it's the inverse function theorem.

The Jacobian is a matrix of the partial derivatives of f, if an elements of f, with respect to different elements of x. So the first row of the Jacobian is the derivatives of the first elements of f, with respect to all the elements of x, and the other rows proceed accordingly. If the determinant of this matrix, evaluated at the solution, is not equal to 0, then this solution is necessarily locally unique. That's the inverse function theorem.

The Jacobian describes the rate of change of this vector-valued function with respect to all of its independent variables. And you may find that, for some solutions, the determinant in the Jacobian is equal to 0. We can't really say what's going on there-- the solution may be locally unique, it may not be locally unique. I'm going to provide you some examples in a second.

And most numerical methods are only going to find one of these local unique solutions at a time. If we have some non-local solutions, that'll cause us problems. So we tend to want to work with functions that have locally unique solutions to begin with.

OK, here's an example. Oh, you have your notes, so you know the formula for the Jacobian. Compute the Jacobian of this function.

So this function has a root at x1 equals 0, x2 equals 0. If you think graphically about what each of these little functions represents, you would agree that that route is locally unique-- it's just one point where both elements of this vector-valued function are equal to 0. Here's what the Jacobian of this function should be.

If you take the derivative of the first element with respect to x1, and then x2, take the derivative of the second element with respect to x1 and then x2-- at the solution, at the root of this function, where x1 is 0, and x2 is 0, the Jacobian is a matrix of zeros. Its determinant is 0. But the solution is locally unique.

The inverse function theorem only tells us about what happens when the determinant's not equal to 0. If the determinant's not 0, then we have a locally unique solution. Solution may be locally unique, its determinant may be equal to 0. Does that makes sense? You see how that plays out?

OK. There's a physical way to think about-- or a geometric way to think about this inverse function theorem. So think about the linear equation, f of x is A x minus b. You can show-- and you should actually work through this-- that the Jacobian of this function is just the matrix A. It says how the function changes, with respect to small changes in x. Well, that's just A-- this is a linear function.

So the equation, f of x equals 0, has a locally unique solution when the determinant of the Jacobian-- which is the determinant of A-- is not equal to 0. But you already knew that, right? We already talked through linear algebra. And so you know when this matrix A is singular, then we can't invert this system of equations and find a unique solution in the first place.

So the inverse function theorem is nothing more than an extension of what we learned about when functions are and aren't invertible. Because a locally unique solution when A is invertible.

In the neighborhood of f of x, in the neighborhood of a root of f of x, we can often approximate the function as being linear. We can treat it as though it's a system of linear equations, very close to that root. And then the things that we learned from linear algebra are inherited by these linearized solutions.

So here's this set of curves that I showed you before. Near this root, let's zoom in-- let's zoom in. These lines look mostly straight. It's like the place where two planes intersect-- intersect this x1, x2 plane-- they each intersect at a line, and the crossing of those lines is the solution. And it's locally unique. Because these two planes span different subspaces.

Here's the case where we may have non-locally unique [INAUDIBLE]. Zoom in on this root, and very close to this root, well, it's hard to tell. Maybe these two planes are coincident with each other, and they intersect and form the same line-- in which case they may not be locally unique. Maybe if I zoom in close enough, I see, no, actually, they have a slightly different orientation with respect to each other, and there is a locally unique solution there. It's difficult to tell here.

So these cases where the curves cross are easy to determine. These are the ones that the inverse function theorem tells us about. These ones are a little harder to work with.

So I mentioned that you can zoom in, and look close to a root, and approximate the function is linear. This is a process called linearization. You are in this for 1-D functions-- f of x, at a point x plus delta x, is f of x plus its derivative times delta x. And this'll typically be valid for reasonably well-behaved functions-- this sort of a linearization is going to be valid as delta x goes to 0. So as long as I haven't moved too far away from the point to x, I can approximate my function in the neighborhood of x using this linearization.

You know, turns out the same thing is true for nonlinear functions. So f of x plus delta x is f of x plus-- well, I need the derivatives of my function with respect to x, and those derivatives are partial derivatives now, because we're in higher dimensional spaces. That's the Jacobian multiplied by this vector of displacements, delta x.

And this will typically be valid as long as the length of this delta x is not too big-- as long as I haven't moved too far away from the point I'm interested in, f of x, this will be a reasonably good approximation. As long as our functions are well-behaved.

There's an error that's incurred in making these approximations. And for a general function that's well-behaved, that error is going to be ordered delta x squared-- and they're in the 1-D, or the multi-dimensional case.

And this sort of an expansion, it's just part of a Taylor expansion for each component of f of x. So we take element I of f. I want to know its value at x plus delta x. That's its value at x. Plus the sum of partial derivatives of that element, with respect to each of the elements of x, times delta x-- each of those delta x's.

Plus, there are some high order term in this Taylor expansion, which is quadratic in delta x instead. There's a cubic term, and so on. These quadratic terms are what give rise to this order, delta x squared error.

In higher dimensions, we typically don't worry about these quadratic terms. We're pretty satisfied with linearization of our system of equations. Sometimes for 1-D nonlinear functions, you can take advantage of knowing what these quadratic terms are to do some funny things. But in many dimensions, you usually don't use that. Usually you just think about linearizing the solution.

So if I know where the solution is, if I know it's close, if I can figure out points that are close to the solution, then I can linearize the function in that neighborhood-- I can find the solution to the linearized equation, instead. That's going to be suitably close to the exact solution. Does that makes sense?

Nonlinear equations, like I said, are solved iteratively. Which means we make a map in our algorithmic map which takes some value xy and generates some new value x plus 1, which is a better approximation for the solution we're after. And we designed the map so that the root, x star, is what's called a fixed point of the map.

So if I put x star in on this side, I get x star in on the other side. By design, the root is a fixed point of the map. The map may converge, or it may not converge, but the root is a fixed point. And we'll stop iterating when the map is sufficiently converged. You guys came up with two different criteria for stopping.

One is called the function norm criteria. I look at how big my function is-- I'm trying to find the place where the function is equal to 0. So I look at how big, in the norm space, my function is for my current best solution, and ask if it's smaller than some tolerance epsilon. If it is, then I say, well, my function is sufficiently close to 0-- I'm happy with this solution. The solution is close enough to satisfying the original equation that I'll accept it, and I stop.

The other criteria, it's called the step norm criteria. I look at two successive approximations for the solution. I take their difference, and ask, is the norm of that difference smaller than either some absolute tolerance, or some relative tolerance, multiplied by the norm of my current solution?

So suppose x is a large number, that spacing between these x's may be quite big, but the relative spacing may actually be quite small. And if the relative spacing is small enough, you might say, well, this is sufficiently converged. And so that's where this relative error, relative error tolerance, comes into play in the step norm criteria.

Suppose x is a small number, close to 0 instead. These steps may be very tiny-- these steps may be quite tiny. They may satisfy this relative criteria quite well, but you may want to put some absolute tolerance on how far these steps are before you stop instead. Because these x's may be small in and of themselves.

And so this one is easy to satisfy, but this one becomes harder. So you usually use both of these. Sometimes you have solutions that are converging toward small numbers, and then the absolute error tolerance becomes important. Sometimes you have solutions that are converging towards large numbers, and so the relative error tolerance becomes important. Does that make sense?

Of course, you can't just use this one, or just use that one. You typically like to use all of these. Because they can fail.

And if the function norm criterion fail-- here's an example where I'm taking some iterations, some approximate solutions that are headed towards the actual root of this function. And at some point, I find that this solution is within epsilon of 0. And so I'd like to accept this solution, but graphically it looks like it's very far away from the root. So this is a case where the function has a very shallow slope.

It's a very shallow slope, and the functional criteria, not so good, really. I call this a solution, but it's quite a ways away from star. So sometimes it's going to work, but sometimes it's not going to work.

Here's the step norm criteria-- here, I have a function nowhere near a root-- I have no idea where I am on this function, I don't know what value this function has, but my steps suddenly got small enough that they're smaller than this absolute tolerance, or they're smaller than the relative error tolerance. I might say, OK, let's stop. I'm not taking very large steps anymore, this seems like a good place to quit.

But actually, my function just after this didn't go to 0 at all, it curved up and went the other way. There's not even a solution nearby. So both of these things can fail, and we try to use both of them instead to evaluate whether we have a reasonable solution to our nonlinear equation or not. Make sense?

Are there any questions about that before you I go on? No. OK.

We also talk oftentimes about the rate of convergence of the iterative process that we're using. We might say it converges linearly, or quadratically. And the rate of convergence is always assessed by looking at the difference between successive-- well, we look at the ratio of differences for successive approximation. So here's the difference between my best approximation, step i plus 1 minus the exact solution, normed, divided by my best approximation at step i, minus the exact solution normed, and raised to some power, q.

And as I go to very large numbers of iterations, i-- this limit should be i, I apologize. I'll fix that in the notes online, but this limit should be i-- is i, the number of steps gets very large. This ratio should converge to some constant. And the ratio will converge to a constant when I choose the right power of q here.

So when this limit exists, and it doesn't go to 0, we can identify what sort of convergence we get. So if q equals 1, and C is smaller than when we say that convergence is linear, what is that saying, really? This top step here is the absolute error, an approximation i plus 1. This bottom step here is the absolute error in step i-- remember two is one for linear convergence. So the ratio of absolute errors, as long as that's less than 1-- I'm converging, I'm moving my way towards the solution. And we say that rate is linear.

If C is 10 to the minus 1, then each iteration will be one digit more accurate than the previous one. The absolute error will be 10 times smaller in the next iteration versus the previous one. That would be great-- usually C isn't that small.

If this power, for which this limit exists, q, is bigger than 1, we say the convergence is super linear. If q is 2, which we'll see is something that results from the Newton-Raphson method, then we say convergence is quadratic. What that means is the number of accurate digits in my solution will actually double with each iteration. Linear, which equals 10 to the minus 1, I get one digit per iteration. Quadratic, I double the number of digits per iteration-- I have one digit on one iteration, I get two the next one, and four the next one, and eight the next one.

So quadratic convergence is marvelous. Linear convergence, that's OK. That's about the minimum you'd be willing to accept. Quadratic convergence is great, so we aim for methods that try to have these higher order convergences. So you really quickly get highly accurate solutions.

You can go back and look at your notes and see that the Jacobian method, and the Gauss-Seidel method both show linear convergence rates. They're linear methods. Does this make sense? OK.

So I meant it mentioned Newton-Raphson. Hopefully, somebody at some point told you about the Newton-Raphson method for solving at least one-dimensional, nonlinear equations. It goes like this, though. You say, I guess my solution is close to this green point here. Let me linearize my function at that point, and find where that linear approximation has a root-- which is this next green point.

And then repeat that process here. I find the linearization of my function, this pink arrow. I look for where that linear function has a root, and that's my next best approximation. And I repeat this process over and over, and it will reliably-- under certain circumstances-- converge to the root.

What does that look like, in terms of the equations? So I linearized my function, so I want to approximate f of x at i plus 1, in terms of f of x at i-- so it's f of x at i, plus the derivative, multiplied by the difference between x i plus 1 and x i. And I say, find the place where this approximation is equal to 0, and determine what the next point that I'm going to use to approximate my solution is. So I solve for x i plus 1, in terms of x i.

How big of a step do I take from x i to x i plus 1? It's this big, so the ratio of the function to its derivative at x i. And the derivative does the job of telling me which direction I should step in. Derivative gives me directionality, and this ratio here tells me the magnitude of the step.

The magnitudes, you know, they're not very good oftentimes, because these functions that we're trying to solve aren't very linear. Usually they're highly nonlinear. What's really helpful is getting the direction right. You could go right, you could go left-- only one of those ways is getting you to the root. Newton-Raphson has this advantage-- it always points you in the right direction. OK?

Of course, you can do this in any number of dimensions, not just one dimension. So you can approximate your function as linear-- f of x i plus 1 is approximately 0. And then let's take our linearized version of the function, and let's find where it's equal to 0. Sometimes what's done is to replace this difference, x i plus 1, minus x i, with an unknown vector, d i-- which is the step size. How big a step am I going to take from x i to x i plus 1?

And so we solve this equation for the displacement, d i. Move f to the other side, so you have Jacobian times d i is minus f. And solve-- d i is Jacobian inverse times f. It's just a system of linear equations. Now we know our step size. So x i plus 1 is x i plus b, or x i plus 1 is x i minus Jacobian inverse times f.

The inverse of the Jacobian plays the same role is 1 over the derivative. It's telling us what direction to step in, in this multi-dimensional space. And this solution to the system of equations is giving us a magnitude of the step that's good, not great, but is taking us closer and closer to the root.

So this is the Newton-Raphson method applied to the system of nonlinear equations. This is really the way you want to solve these sorts of problems. It doesn't always work-- things can go wrong.

What sorts of things go wrong here? Can you guess? Yeah?

AUDIENCE: [INAUDIBLE]

PROFESSOR: OK, this is good. So in the 1-D problem, sometimes the Newton-Raphson method can get stuck. So it won't have good necessarily global convergence properties. If you have a bad initial guess, it might get stuck someplace, and the iterates will converge. That can be true. What else can go wrong?

AUDIENCE: [INAUDIBLE]

PROFESSOR: Good, so if you're derivative is 0, that's going to be problematic. What's the multi-dimensional equivalent of the derivative being 0?

AUDIENCE: [INAUDIBLE]

PROFESSOR: What's that? Singular Jacobian, right? So if this is J, the Jacobian, has some null space associated with it, how am I supposed to figure out which direction to step in? There's some arbitrariness associated with the solution of this system of equations. So the derivative is 0 in the 1-D example, that's a problem.

That problem gets a little fuzzier, but it's still a big problem when we try to solve for the step size, or the step-- the Newton-Raphson step. They may not be able to do this.

You don't run into this very often, but you can, from time to time. One place where this is going to happen is if we have a non-locally unique solution. When we have one of those, we know the determinant of the Jacobian at that point is going to be 0. If we're close to those solutions, well the determinant of the Jacobian is going to be close to 0-- you might expect that the system of equations you have to solve becomes ill-conditioned.

So even though there may be an exact solution for all the steps leading up to that point, the equations may become ill-conditioned. You may not be able to reliably find those solutions with your computer, either. So then these steps you take, well, who knows where they're going at that point. It's going to be crazy.

There are ways of fixing all these problems. Let's do an example.

This is a geometry example, but you can write it is a system of nonlinear equations, as well. So we have two circles-- circle f 1, circle f 2 in the x1, x2 plane. They satisfy-- these are the locus of points, that satisfy the equation f 1 of x 1 and x 2 equals 0-- this is the equation for one circle, and this is the equation for the other circle. And we want the solution, we want the roots of this vector-valued function, f, for vector-valued x. And those are the intercepts between these two circles.

You can do it using Newton-Raphson, so you're going to need to know the Jacobian. So compute the Jacobian of this function. This is practice-- maybe most of you know how to compute a Jacobian, but some people haven't done it before. So it's always good to make sure you remember that the first row of the Jacobian is the derivatives of the first element of f. And later rows are later elements.

You don't want the transpose of the Jacobian. Then it won't work.

OK, so should look something like this. There's your Jacobian. The Newton-Raphson process tells us how to take steps from one approximation to the next. The step is equal to minus the Jacobian inverse evaluated, and my best guess for the solution multiplied by the function-- evaluated at my best guess of the solution.

You're never going to compute Jacobian inverse explicitly-- that's just code for solve this system of linear equations. So use the slash operator in MatLab, for example.

And here you go-- I had an initial guess for the solution, iterate 0 at minus 1 and 3. This is somewhere outside the circles-- it's pretty far away from the solutions. But I do my Newton-Raphson steps, I iterate on and on. And after four steps, you can see that the step size in absolute value is order 10 to the minus 3.

The function norm is order 10 to the minus 3, as well-- and maybe order 10 to the minus 2, but it's getting down there. These things are decreasing pretty quickly. And I move to a point that you'll see is pretty close to the solution.

Here's some things you need to know about Newton-Raphson, you'll want to think carefully about as we go forward. So it possesses a local convergence property. I'm going to illustrate that graphically for you.

So here, I didn't solve the problem once, I solved it-- I don't know, 10,000 times. And I chose different initial points to start iterating with. Here, minus 1, 3, that was one point. But I chose a whole bunch of them. And I asked, how many iterations-- how many steps did my Newton-Raphson method have to take before I got sufficiently close to either this root here, or this root there?

I don't remember what that convergence criterion was-- it doesn't really matter, but there was some 10 to the minus 3, or 10 to the minus 5 or 10 to the minus 8, the convergence criterion that I made sure the Newton-Raphson method hit, in both the function norm and step norm cases. And then, if the color on this map is blue-- the solution converged to this star in the blue zone-- if the color's orange, the solution converged to the star in the orange zone. And if the color is light, it didn't take so many iterations to converge. And if the color gets darker, it takes more and more iterations to converge.

So that's the picture-- that's this map. I solved it a bunch of times, and then I mapped out how many iterations did it take me to converge to different solutions? So you can see if I start close to the solution, the color is really light. It doesn't take very many iterations to get there. And the further way I move in this direction-- still doesn't seem like it take so many iterations to get there, either.

I need a good initial guess-- I want to be close to where I think the solution is. Because once I'm over here somewhere, I do pretty well. And the same is true on the other side, because this problem is symmetric.

There's a line down the middle here, and along this line, the determinant of the Jacobian is equal to 0.

So we talked about these points with the Newton-Raphson method. And if I pick initial guesses sufficiently close to this line, you can see the color gets darker and darker. The number of iterations required to converge the solution goes way up.

Now, the Newton-Raphson method possesses a local convergence property. Which means, if a locally unique solution, there's always going to be some neighborhood around that solution for which the determinant of the Jacobian is not equal to 0. And in that neighborhood, I can guarantee that this iterative process will eventually reach the solution.

That's pretty good. That's handy. These iterates can go anywhere-- how do you know you're getting to the solution? Are you going to waste your time, or are you going to get there? So that's this local convergence property associated with it, which is nice.

But it'll break down as I get to places where the determent of the Jacobian is equal to 0. so there could be a zone-- it's not in this one-- there could be a zone, for example, like a ring on which the determinant of the Jacobian sequence is 0. And if I take a guess inside that ring, who knows where that solution is going to go to. Could be something like Sam said, where the iterative method just bounces around inside that ring and never converges.

But when I have roots that are locally unique, and I start with good guesses close to those roots, I can guarantee the Newton-Raphson method will converge. I'll show you next time that not only does it converge, but it also converges quadratically. So you start sufficiently close to the solution, you get to double the number of accurate digits in each iteration. You can see that happening here.

OK, so I have one accurate digit, now I have two. The next iteration I'll have four, and so on. The number of accurate digits is going to double at each iteration.

So that's going to conclude for today. Next time, we'll talk about how to fix these problems with the Newton-Raphson method. So there are going to be cases where the convergence isn't ideal, where we can improve things. There are going to be cases where we don't want to compute the Jacobian or the Jacobian inverse. And we can improve the method. Thanks.