PROFESSOR: So today we're going to talk about partial differential equations. Let me just give a quick review where we are. So we're going to have a PDE lecture today. Then Professor Swan is going to do a review on Friday. There's no lecture on Monday, but you have the quiz, too, Monday night. Wednesday, next week is Veterans Day so it's a holiday. There's no classes at MIT. So the next time I'll be lecturing to you will be a week from Friday. The homework following the quiz will be posted and so you guys can get started on that.

That homework involves a COMSOL problem which, for those of you who had trouble with COMSOL, might take some extra time, since it's the first time you're doing it. So I would suggest you at least give it a try early. I'm sure the TAs would be very happy to help you to force COMSOL to cooperate. Bend COMSOL to your will.

All right, so we're going to talk about partial differential equations. And, as it sounds, what's different about a partial differential equation is, it has partial derivatives in it. And so we have a lot of equations like this. So for example, I work a lot on this equation. It's a poor choice, isn't it?

OK, so this is the conservation equation for a chemical species in a reacting flow. So the change in the mass fraction of the species z, at a point, xyz at time, t. That differential is given by a diffusion term, a convection term, and a reaction term. So this is a pretty famous equation. This is part of the Navier-Stokes equations for the whole reactive flow system, which I hope you guys have seen already, and for sure you will see, if you haven't seen them yet.

And there's a lot of other partial differential equations. I work a lot, also, on the Schrodinger equation. That's another partial differential equation. And what's special about the partial differential equations is that, in this case, this partial derivative is respect to time, holding all the spatial coordinates fixed. And these partial derivatives are respect to space, holding time fixed.

A very important thing about partial derivatives, which you probably have encountered in 1040

already, is, it really depends what you hold fixed. And you get different results if you hold different things fixed. Now the implication, when you write an equation like this, is that whatever partials you see-- When I wrote this, if I have this and it has a term that's like d squared, dx squared, is one of the terms in this thing. This says t. This says x. The convention is, oh boy, I better hold x fixed. And here must be, I must hold t fixed. Because there's another one in the same equation.

Now, you can change coordinates however you want. Just really watch out that you carefully follow the rules about what you hold fixed, when you change things. Otherwise you can cause all kinds of craziness. Now you guys probably took multivariable calculus at some point in your distant past. And they told you very carefully what the rules were. So follow them. And I suppose, when you're doing thermo right now? Have you started to do all this partial derivative stuff? Yeah, so you've seen how completely confusing it can be with negative signs showing up all over the place. And other things like this. I don't know if you've encountered this yet? At least, I thought it was confusing when I took it. So just be aware. It's the same thing.

Now, often you do want to change the equations. So you can write down an equation like this. But, for example, if you have a special symmetry of the problem, like it's cylindrical, for example, then you might want to change the cylindrical coordinates. And then you have to just be careful that you write that correct Laplacian and cylindrical coordinates. Make sure it doesn't mess up anything about what you're holding fixed.

Sometimes you might want to be crafty and use some weirdo coordinates. For example, if you wanted to solve the Schrodinger equation for h2 plus, it turns out there's a special coordinate system called the elliptic coordinate system. And in that coordinate system, the partial differential equation is separable. And so, if you do it in that coordinate system, you can actually solve analytical solution, which is pretty unusual for the Schroedinger equation.

But it's a pretty goofball coordinate system and you really have to watch out, when you convert, to make sure you do it just right. But it's just the rules, chain rule and the rules about how to keep track of what's being held constant. And when you change, will tell them, how, being held constant, how to change it.

Now in principle, this problem is like no different than the ODE BBP problems we were just doing. You just have a few more terms. And so, you can use finite element. You can use finite differences. You can use basis set methods. And it's really no different. So all those methods

basically look the same. You just get equations that have more unknowns in them. Because, for example, if you're using finite element methods or basis set methods with a local basis set, you need to put points both in the x direction and the y direction, and in this case, the z direction, and then maybe also in the time direction. So now you have a lot of mesh points in all different directions. And so you get a lot of mesh points. And so fundamentally, there's no problem, but in practice, there's a big problem if the number of unknowns gets to be so large. That's the main problem.

So think about it. You have some kind of equation like this. It's in three spatial dimensions and in time. How many points do you think you need to discretize your solution in the spatial coordinate? There might be somewhere, might be 100, I don't know, points might be enough. And so now I have x and y and z. So I'll need, well, just in COMSOL, you saw this on Monday. Suppose I just have two dimensions. Suppose I just have x and z. And I start putting points in. Well, I want to have some points along this way. And I want to have some points along this way. And so, I actually have an unknown here and an unknown here, and one there and one there, and one there and one there. There's quite a few of them.

I think of how many points I have. Each of those is a point where I have a little basis function. And each one of those has a magnitude of how much weight I have on that basis function, things they called the d's before in the basis functions. And I have a lot of them. Now, it's going to be, if I have 100 points this way and 100 points this way, I have 10,000 of these little points. That means I have 10,000 unknowns I have to solve for. 10,000 unknowns are a lot harder to solve for than 100 unknowns. and if you saw, on some of the problems we did with the ODE BBP problems, even with 100 unknowns, we were having a lot of trouble, sometimes, to get the real solution. All right? So you can just see how it just gets to be, like, a big numerical problem.

So that's two dimensions. The third dimension, you'll get another 100 times as many points. You might have a million unknowns. And once you get up to a million unknowns, we're starting to talk serious math, now. OK? Even for your good computers that you have. In addition, it's not just you have the number of mesh points, but it's the number of mesh points times the number of variables at each mesh point. So in the Navier-Stokes equations, what variables do you have?

AUDIENCE: [INAUDIBLE]

PROFESSOR: Louder.

AUDIENCE: [INAUDIBLE]

PROFESSOR: Velocity and pressure, OK. So you have pressure. We have the different components of velocity. What else we got?

AUDIENCE: [INAUDIBLE]

PROFESSOR: Sorry, what?

AUDIENCE: [INAUDIBLE]

PROFESSOR: Yeah, though, yeah. Density or temperature. Yeah, temperature maybe, so. So maybe temperature would be what you might use. And then you'd have a state function maybe with these guys.

What else would you have? So then you'd have all these species, right,? The mesh fractions are all the species. So you have, like, y, species one. Mesh fraction, species two. However many you got. So that's a pretty big set of variables and you need to know all these numbers at each point. So this might be-- how many have we've got? One, two, three, four, five, plus however many species we have? So this is n species plus 5 times the mesh. And I told you the mesh was what, a million? So maybe I have 10 million unknowns.

So the issue, to a large extent, has to do with just, can I solve it? Can I solve a system of 10 million unknowns? 10 million equations, 10 million unknowns. All right, so that's going to be one big issue. Now the other issues, we still have the same problems we had in ordinary differential equations, that we have to worry about numerical stability. We have to worry about actually, is the whole problem even stable to begin with? Physically? Because if it's not stable physically, then probably we're not going to get a stable solution.

So you have all those kinds of problems that we had before in the ODE case, and now it's just sort of amplified by the fact, now we have boundary conditions in more than one dimension. So we have multiple chances to screw it up. And we're trying to figure out how to set the boundary conditions correctly. And we have to worry about stability in all the different directions. So those are the different kinds of things we have to worry about. But it's all the same problems you had before, just amplified.

Now, because of this problem of the very large number of unknowns, currently, if you have a problem that has two dimensions, you can kind of very routinely solve it. And things like COMSOL on your laptop, you can usually do those. When you get up to three dimensions, you might be able to solve it or might not be able to solve it. And it could be, like, a really big challenge sometimes to solve it. And then in reality, we have problems like the Navier-Stokes is four dimensions, right? There's time, as well. So then that gets to be really hard. And then the Schrodinger equation has three times the number of electrons dimensions in it. So that gets impossibly hard.

And so, specialized methods are developed to handle those systems with a lot of dimensionality. So, like, for the Schrodinger equation, people almost always do it with basis set methods. And there's been a huge effort, over decades, to figure out really clever basis sets that are really close to the real solutions, so that you don't need to use too many basis functions to get the good solution. So that's the way to keep the number of variables down.

In the Navier-Stokes world, I'll show you what people do to try to deal with that. There's many other problems. I don't know all the problems in the world. But in general, when you have three or more dimensions in your PDE system, you're looking up special tricks. You're looking up, how do people in this field deal with this particular PDE? Special ways that are good for that kind of PDE.

All right, so I said, one problem was that you could have, that the PDE system, itself, can be intrinsically unstable. So one example I know like that is, detonation problems. So if I have a system that has some explosive in it, or a mixture of hydrogen and oxygen even, and I have it in a tube or something, if I change my initial conditions, I can go from the case where it's stable, that it just sits there, a hydrogen and oxygen. Or it could turn into a regular burning. Or if I change the conditions a little bit differently, it could change into a detonation, where it actually sends a shockwave down the tube faster than the speed of sound, totally different situation.

So those kind of systems can be really sensitive to the physical initial conditions. A very small spark can make a really big difference in a system like that. And also, in your numerical solution, if the thing is physically sensitive, it'll typically be sensitive also to numerical noise. If you get numerical noise at some point, that might be enough to kick it over from, say, not igniting at all to having a detonation, which is a completely different solution.

But another case, like in biology, if you're trying to study the growth of a cell culture or the growth of a tumor, as you know, if somebody has a tumor and one of the cancer cells mutates and does something different, it can have a really different outcome of what happens to the patient, what happens to the tumor. The whole morphology can change completely. Everything about it can change. So that's the kind of system, also, that can be very sensitive to small details. A small change in one cell, for example, could be a really big difference. So this is not just in combustion problems you have these kinds of instabilities. All kinds of problems have this kind of thing. You can have just regular chemical reactors, like in multiple steady states, little tiny fluctuations can make it jump from one steady state to another one. That's another very famous kind of problem.

There's another class of PDEs where it's stable in some sense, but it causes a problem. We call those hyperbolic. And those are like wave equations. The solutions are propagating waves. So the equations of acoustics, the equations of electromagnetism, are wave equations where, basically, a wave propagates more or less without any damping. So what happens, then, is you have some numerical noise that will make a little wavelet that will propagate, more or less, without any damping, too. And it'll keep bouncing back and forth inside your domain numerically. And if you keep introducing numerical noise, eventually the whole thing is just full of noise. And it doesn't have much relation to the real physical solution.

So hyperbolic systems of differential equations need special solution methods. That in the numerical method, it's carefully trying to damp out that kind of propagating wave that's coming from the noise. And they set them up a special way. I'm not going to talk about how you solve them in this class, but there's a special, whole group of solution methods that people use who are solving acoustics equations and solving electromagnetic equations. If you get into solving shockwave equations-- I actually work for Professor [? Besant. ?] He has, like, electrochemical shocks in some of his systems. You have the same kind of thing. You have to use special mathematics to correctly handle that, special numerical tricks.

But we're not going to get into that in this class. If you get into those things, you should take the PDE class for hyperbolic equations and they'll tell you just all about the solvers for those kinds of things and special tricks. And there's journal papers all about it.

Here, what we're going to focus primarily on two kinds of problems: elliptic problems and parabolic problems. And those problems are ones where there's some dissipation that's very significant. And so, in an elliptic problem, you introduce some numerical noise and as you

solve it, the numerical noise kind of goes away. Sort of like intrinsically stable all throughout. Now the real definition of these things, parabolic, hyperbolic, elliptic, has to do with flow of information. So in a hyperbolic system, information-- there's regions of the domain that you can make a change here and it doesn't make any effect on some domain part over there. And so that causes-- because that is the real situation, that also propagates into the numerical solution method.

So for example, if I'm modeling a shockwave that's moving faster than the speed of sound, I can introduce any pressure fluctuation I want behind the shockwave, and it has no effect on the shockwave. Because the pressure waves from that pressure fluctuation are traveling at the speed of sound. They won't catch up to the shock. So they will have no effect whatsoever. So if I was judging what was happening by looking at what's happening in the shock, I can introduce any kind of random noise I want back over, downstream of the shock, I guess. Yeah? And it won't make any difference at all. And so, that's part of the reason why I need a special kind of solver.

If I have elliptic equations, every point affects every other point. A famous case for that is, like, this steady state heat equation. You have some temperature source here, a heat source here, and you have some coolness source over here. And there's some heat flowing from the hot to the cold. And the heat flow is kind of slow enough that everything sort of affects everything else, since you actually have, the heat is trying to flow by, sort of, every path. And there's some insulation that's resisting its flow. And it has been in the connectivity of flow a certain way. And those kinds of equations are called elliptic. And the methods that we use for the relaxation mesh that we showed are really good for those kinds of methods, for those kind of problems.

And then another kind of problem that we get into a lot is like this one. This is called parabolic. And typically, in the cases we see that are parabolic, we have time as part of the problem. Always what we think should happen is that what happens in the future depends on what happened in the past, but not vice versa. So you would expect that when you're computing what's happening at the next time point, it's going to depend on what happened at earlier time points. But if you try to do it the other way around, it would be kind of weird, right? You wouldn't expect that what in the future really affected stuff in the past.

So it naturally leads us to sort of pose it as an ODE IVP, or IVP kind of problem. And we know initially, some time zero something, and we're trying to calculate what happens at some later

time. And so those kinds of problems we call parabolic. And the methods that we're going to talk about mostly are ones for solving parabolic and elliptic problems. And you can read, in the middle of chapter 6, has a nice discussion. Just has the mathematical definitions of what these things are.

Now, even in a steady state problem, the problem can have a sort of directionality to it. So if you have a problem that's very convective. So you have a flow in a pipe. And you have a nice, fast velocity flow down the pipe. What's happening downwind is very much affected by what happened upwind, but not vice versa. So maybe you're doing some reaction upwind. Say you're burning some hydrogen. You're going to get water vapor made upwind and it's going to flow downwind. And you're going to have steam downwind.

If I squirt a little methane in downwind, it doesn't actually do anything to what happened upwind. I still have a hydrogen and oxygen flame up there and it's been burning and making steam. Do you see what I mean? So it has a directionality, even though I might do it as a steady state problem and not have time in it at all. And you've probably already done this, where you can convert, like, a flow reactor. You can write it, sort of, as a time thing or as a space thing. Time and space are so related by the velocity of the flow. We'll have similar kinds of phenomenon then. It's almost like a parabolic time system that, if the flow is fast enough, the fusion backup, anything moving upstream is negligible. It's all moving from upstream to downstream.

And so then, you'll have the same kind of issues. In fact, you may even try to solve it by just starting it at the upstream end and competing stuff and then propagating what that output from that first one does to the next one downstream, and then do them, one at a time. That would be one possible way to try to solve those kinds of problems that are highly convective. As you slow the velocity down, then the diffusion will start to fight the velocity more and more. If you make the velocity very slow, the Peclet number very low, then you really have to worry about things diffusing back upstream. And then it's a different situation. And that would be more like the elliptic problems we have. Because things downstream would actually affect upstream, if the flow is really small.

But typically for our problems, the velocity is always really high. Peclet numbers are really large and so not much goes from downstream to upstream. It's almost all upstream to downstream.

This leads to a famous situation. If you look in the textbook at figures 6.7, 6.8, it's kind of a very famous problem. That if you just do this problem. So this is just an ODE problem. So I just have convection and diffusion, no reaction, very simple. This problem. You'd think that I could just use my finite differences, where I would put, I could use the center difference for here and here and change this into this obvious looking thing.

You think that that would be a reasonable thing to do, right? And you can do the same one here. You can write-- I'm terrible with minuses, sorry. So these, the finite difference formulas. And so, you can write these like this. This stays equal to zero. And now it's just an algebraic problem to solve. A system of equations, you have equations like this for every value of n. And there's the mesh points. You guys have done this before, yes? Yes.

OK, so famous problem is, if you actually try to solve it this way, unless you make delta z really tiny, you get crazy stuff. So on figure 6.7, they actually show what happens for different values of delta z, which is different values of what they call the local Peclet number. So there's a thing called local Peclet number, which is-- OK? If you make the local Peclet number too large, actually anywhere bigger than 2, and you try to solve this equation, what you get is oscillations, unphysical oscillations. They'll make the phi go negative. It's crazy. It looks like the simplest equation in the world, right? It's a linear equation, so what's the problem with this? But it doesn't work.

And so, then you could think, well, why doesn't it work? And there's, like, multiple ways to explain this about why this doesn't work. I think that the best way to think about it is about the information flow. So if the local Peclet number is large, that means that phi is just flowing downwind and diffusion is not doing much. Because the Peclet number is big. And so, you really shouldn't use this formula to calculate the flow. Because what happens downwind, at point n plus one, I guess. Here we have the flow this way. So here's n minus one. Here's my point n, n plus 1.

I really should not include, is this right, n plus 1 in this formula. Hope I have those signs right. I apologize if I had them backwards. Because I don't think that what happens downstream should really affect this at all. So really, I would do better to change from this center difference formula to what's called the upwind difference formula. Where they would say, OK, instead, let's use phi of n minus, the phi of n minus 1 over delta z. Just use that instead of using this formula.

Now, you wouldn't think that that would make much difference. But it makes a gigantic difference. And so if you look at figure 6.8 in the textbook, you see you get perfectly stable solutions when you do that. But where you do this one, you get crazy, oscillatory, unphysical solutions, unless you choose delta z really tiny.

Now, there's other ways to look at this. So I think this is the correct way, the best way to think about it is, it has to do with information flow. If you try to make your solution depend on stuff that it doesn't really depend on. So upwind does not really depend on downwind. If you force it to, by choosing to include that information here, you end up with crazy stuff. So that's one way to look at it.

Another way to look at it is that, if you really want to compute these derivatives, you've got to keep delta z small. Because we're taking a differential. We're turning it into a finite difference. And if you take your delta z too big, then it's a terrible approximation to do this. And you can look at it. You can see that what we're doing is, when we have this problem where the local Peclet number is getting too large, is we're actually choosing delta z bigger than the, sort of, the thickness of the solution. So if you look at the solution, the analytical solution of this problem, it's like this. Where this has a very, very thin layer at the end. It's like the flow has pushed all your stuff to one side.

And if you choose your delta z to be from here to here, that's how big your delta z is, and then you're computing the derivative of this point halfway along from here to where this is. You compute your derivative like that. It's not a very good, not a very accurate representation of what the real derivative is here. And in fact, you really should have a bunch of points here and here and here and here and here to really resolve the sharp edge. And so you did something really crazy to use a big value of delta z to start with. So that's another way to look at this problem.

Now, the fix, by doing this upwind differencing, I think the best way to look at this is saying, well, I'm just going to make sure I only depend, I only make the equations depend on what's upwind, with the convective term. Because there's nothing convecting from downwind. So just leave that out. So that's a conceptual way to think about it.

Another way to think about it, which is in the text, is that by doing this, you're introducing another thing called numerical diffusion, where you're actually, effectively increasing the diffusivity. Because this is a very poor approximation to the differential. This is a asymmetrical

finite difference formula. It's not a very good value estimate of the derivative, if delta z is big. But you can write it out carefully and write this out and say, oh, this actually is sort of like this plus an effective diffusivity chosen very carefully, so the terms cancel out just right. And it turns out, if you look at it that way, the effective diffusion you've added, by using this instead of that, is just enough to make the local Peclet number, if you compute it with the effects of this d effective, to stay less than 2. Which is what you need as a condition to make this numerically stable.

So I strongly suggest you read that. It's only two pages long in the textbook. It's very clear. Just read, just look at it if you haven't seen this before. There's a similar thing and it might be relevant to the COMSOL problem that Kristyn showed you on Monday, is you can-- In that case, there was a region of the boundary here that had some concentration c naught, and then there was, over here, the concentration was zero. The boundary is zero. Here, it's some number, 17, whatever number it was.

And if you think of how to model what's going on, one way to look at it is, I have a diffusive flux coming in from the drug patch, diffusing the drug into the flow. And I have this big, giant flow, flowing along here. And the flow gets slower as I get closer to the wall because of the friction with the wall. And so if I looked somewhere right in here, I could just draw a little control volume, and look what's happening. I have a diffusive flux coming in here. I don't have any drug up here, so there's no drug coming in here. But I have drug leaving. So basically, it's coming up and it's making a right-hand turn.

And so then I could think, well, if I was going to try to figure out, what's the steady state concentration of drug in there, I could say, well, there's a certain amount of drug entering from the diffusive flux, which would be just this times dcdy. This is the y direction. And then this part over here would be the velocity, actually, just times the concentration I have in here, right? So how much is flowing out. And it's probably, my units are screwed up so there's-- Well, maybe not. That's OK. Is that all right? Units are screwed up. You guys will get it right. It's probably an area. Something to do with this size right here. This one here, too. Is that right?

So you could compute what the steady state concentration would be if it's just coming in and flowing out. And so, this is the finite volume view of this kind of problem. And so you can write the equations that way, instead. And what you really care about are what the velocity flows and the diffusive fluxes are on these boundaries around the point. So you don't try to compute things right at the point. You compute around it.

And one way to look at this is, we just did finite differencing. We were looking at what was happening coming in. And then, this is actually a center difference around this interface, which is halfway between the two points. And so actually, this is a good approximation for the finite volume point of view. This is all right?

Now, these are all different ways to write down the equations. All of them are correct in the limit that delta z goes to zero. If you have an infinite number of mesh points, all these are exactly the same. But they lead to really different numerical properties when delta z is large. And they're all inaccurate when delta z is large, so it's all about what happens with the inaccuracies.

Now, we can go ahead and take any problem, even really complicated 3-D problems or 4-D problems like this, and we can write them with relaxation methods. And what we're basically doing in a problem like this is turning it into some function of y is equal to zero, where y is the value of all the unknowns. And there's a lot of equations. So I might have 100 million equations and 100 million unknowns that are the values, or every state variable at every state point at all times. And I can write it down, by finding differences, by finding elements, by finding volumes. Any way I want, I can write down an expression like this. And I know in the limit, as I make the number of elements of y go to infinity, it will actually be the real solution.

And the problem is, my computer can't handle infinite number of unknowns. In fact, it's even going to have trouble with a 100 million unknowns. And so, I have to figure out what to do. So how would I solve this normally, is I would take the Jacobian of this and I would do, I'd take an initial guess and I'd update it and have the Jacobian times my change from the initial guess to the equal negative of f, right? You guys have done this a million times.

As Newton-Raphson is how you'd find improved from an initial guess. You have to provide some initial guess. This is actually pretty hard if you have 100 million unknowns. You have to provide the value of the initial guess for every one of the 100 million. But somehow you did it. And now you want to refine that guess and this is the formula for it. And so, you just use backslash, right? But the problem here is that now f is a 100 million long vector and this is a 100 million squared matrix. And a 100 million squared is pretty big, 10 to 16th elements in it. So I'm not going to just write that matrix down directly like this.

Now, we cleverly chose local basis functions, so this is going to be sparse, this Jacobian. So most of the numbers, are those 10 to the 16th numbers in this matrix are zero. So we don't

have to represent them. But there still might be quite a few. Probably the diagonal events are non-zero, so that's like 10 to the 8th of them right there. And we've got, probably, a couple of other bands. So I don't know, might be 10 to the 9th non-zero elements inside this thing, which is pretty nutty. And depending on how much memory they provided in the laptop they gave you, you might be filling up the memory already just to write down j.

And certainly, if you tried to do Gaussian elimination to solve this, you're going to have a problem because you get what's called fill in. Do you guys remember this? Even if you start from a very sparse Jacobian, as you run the Gaussian elimination steps the number of non-zero entries in the intermediate matrices gets larger and larger. Remember this? And so, even if, initially, you can write down j and store it in your computer, you could easily overwhelm it with the third iteration or something.

So you're not going to just use Gaussian elimination. So what do you have to do? You want to find a method. They're called direct. Direct methods to solve this kind of problem, in order to get your increment, which is going to be your update, to get a better estimate of what the solution is. And the direct methods are ones that you don't actually store the gigantic matrix. So you're trading off CPU time for memory. So you don't have enough memory to store the whole thing. Which we, normally you would do this with Gaussian elimination and huge steps. It would solve it perfectly, right? Right, Gaussian elimination's great, backslash. You guys have used it a few times. It's a really nice program. It always works. It's really nice.

But you're going to give up that because you can't afford it because the memory, it's consuming too much memory and your cheapo department didn't buy enough RAM in your computer for you. So you're going to instead try a direct method which is, so it's trading off CPU time versus RAM. So you're going to reduce how much memory you actually actively have, but you're going to expect that it's going to cost more CPU time. Because otherwise, everybody would do something else besides Gaussian elimination all the time, and they don't.

All right, so what's the method we're going to use? It turns out that variance on the conjugate gradient method turned out really well. So we mentioned this earlier, briefly. What you do is, instead of trying to solve this, you try to solve this.

So you just try to minimize that, right? You know at the solution this is going to be zero. Right, jy plus [INAUDIBLE] zero when this is solved. So you try and vary and find the delta y that makes this go to zero by minimizing it. And then this method, it turns out that the conjugate

gradient, if everything works great, this is guaranteed in n iterations to go directly to the solution of this kind of problem. Now n is large, now, because we have 10 to the 8th unknowns. So that's 10 to the 8th iterations. You do anything in 10 to the 8th iterations, you're not really sure it's really going to work. So that's a warning. But at least it should get, even after a few iterations, you should get a smaller value of this than you had before.

And so this is the method. And what's really nice about this method, if you look into the details of how it works, it never has to store any intermediate matrices. So all it needs is the capability to multiply your matrix times some vector. And from that multiplication, it figures out a step direction. That's the trick of this method. And because it only has to do a forward multiplication, you don't have to actually store j. You can just compute elements of j, as needed, in order to evaluate this top product. This j times v. And you don't have to ever store the whole j at once. So it's really great for RAM to do this method.

Now, as I said, when you do 10 to the 8th iterations, things don't always work out so well. And so people have found that when you go more than maybe 10 or 20 iterations like this, you tend to pick up numerical problems. And so, they've worked out better methods. And there's one that people use a lot for these problems called bi cg stab. Which is biconjugate gradient stabilized. The applied mathematicians went crazy trying to figure out better methods like this. And this is the one that, I think currently, people think is the best. Though probably, there's a lot of research on this, so maybe there's even better ones now. But inside Matlab, I think this is the best on they have.

And this is a method for doing this. Now, it's iterative. Now, let's remember what we're doing. We're trying to solve this problem. We started with a guess. We're going to break it up into an iterative problem like this, where we're going to do some steps, delta y. This is now doing iterative procedure in order to compute delta y. So we have an interim procedure and another iterative procedure inside the first iterative procedure. So this is going to be a lot more CPU time. So just to warn you. And it's also, it's not guaranteed like Gaussian elimination, to beautifully come to machine precision solution. It's going to get you something. But anyway, this is what people do. So that's one good thing to try.

Now, this whole approach is sort of based on Newton-Raphson. We know that doesn't have the greatest radius of convergence. So you need to provide a pretty darn good initial guess. So how are we going to get a good initial guess if you have a 100 million unknowns? So what methods do we know? So one idea is you could do things like homotopy, stuff like that. If you

could somehow take your PDE system, get rid of the nonlinear terms. You're really good at solving linear systems of PDEs. You start with that and then you gradually turn the non-linear term on. Maybe you could coax it over. So that's one possibility, if you're really good at PDEs.

Another possibility that I've run into a lot is the problem you're trying to solve, for example, might be a steady state problem like this. Where this is zero and you try to figure out what the steady state situation is in a flow reactor, for example. And so, it's actually, your steady state problem came from a time dependent problem. And if you think that your time dependent problem really converges to the steady state problem, sort of without much concern about what the initial guess is, then you could put any random initial guess in and march it along in time enough, and eventually, it should end up at the solution you want.

So that's the idea. So that's called a time marching idea. So there's one idea, sort of the homotopy continuation kind of idea. That's one idea about what to do for initial guesses. Another idea is the time marching idea. And the key about it is, you have to really believe that, physically, the system does march to the steady state you want. If it does, then this is a good idea. If it doesn't, then you're not going to end up where you want, because it's not going where you want. And so, you have to watch out.

But in some situations, this is true, that things work. So I do a lot of combustion problems. A lot of these things, you light a spark on a stove burner. No matter how you light it, it ends up basically the same flame. So that's a good one. But I have some other ones where I light a candle and I'm in a convective field, a turbulent field, and it flickers. And no matter what, it always flickers. And I never get a steady state, so I'm never going to find a steady state solution with that one. And so I could time march until I retire. My computer's still burning CPU time all that time and it'll never get to a steady state solution. So you really have to know what the real situation is. But if you have one where it's going to converge, then this is a reasonable idea.

Now let's talk about the time marching. The time marching, you have, say, dy, dt is equal to some f of y where this is the discretized spatial vergence. I took all the space dimensions and replaced them with the finite difference expressions for the derivatives. Or I did colocation or something to convert this into algebraic equations. So the right-hand side is now algebraic equations, no longer differentials. And now, I'm going to time march this and I just-- This is very long. It's a 100 million.

So how are we going to time march that? Well there's kind of two schools of thought for this. One school of thought is, if I can use an explicit method, an explicit ODE solver, those are pretty nice. Because all I have to do is evaluate f and then I can march along and I never have to save any matrices. Because remember, look at those formulas, y, y new, by the explicit method, is equal to some function g of y old.

And this is the update formula. It can be forward Euler. It could be RK 2. It could be whatever. But there's just some explicit formula. I plug in the old vector. I compute a new vector. I never had to do any inversions of matrices or anything. So that's good. And this is, indeed, the way that the Navier-Stokes equations are solved currently by the best possible solver. So the best solver in the world discretizes in space and then does ODE IVP using explicit method for marching it.

They do problems that are pretty big. So they'll have like 100 million mesh points and then 100 species. So they have 10 to the 10th unknowns. And at each time step, you're computing another vector that's 10 to the 10th long. So it's a pretty big problem. And they use, like, 40% of one of the national supercomputer centers will be running at one time for one job like this. But it works. You never have to store anything that's too big. The biggest thing you have to store are the vectors. Which, you're always going to have to store some object the size of your solution, right, if you're going to get your solution. So that's one possibility.

Now, this is really good if an explicit solver can solve it. So that means it's just not stiff, your problem. And also, if you want a time accurate solution. So in that case, they actually want the time accurate solution. But suppose we're trying to really solve the steady state problem over here. Then we really don't care about the time accuracy. We're not going to report anything about our time steps. In the end, we're only going to report our steady state solution. That's all we cared about. So in that case, there's no reason to try to have a really high accuracy, explicit formula and try to make sure we all our time steps are exactly right. We're just going to throw away all those time points anyway. We'll just keep the final one as our initial guess for a Newton-Raphson solve to find the steady state.

So in those cases, you might instead want to use-- So this is time accurate. There's another method. I don't know if I should call it time inaccurate. If you don't care, if the solution is really, what the solution yft is, all you care about is where you get to, then you can do other methods. And one of them is the backward Euler. So you can say, well, y new is equal to y old plus delta t times f of y new. Now, the advantage of this is, I can make delta t pretty large. This is

guaranteed stable as long as the ODE system is stable. Remember? And so, this will go to this, go to the solution pretty well, to the steady state solution.

It won't go there in the right amount of time because this is a very low order of formula. It's not really accurate. But it will end up to the real steady state solution. This is what's actually used a lot. However, the problem with this is, this is now an implicit equation. So we may have to solve it with some method like Newton-Raphson. But we got into this because we couldn't solve our Newton-Raphson steps because we didn't have a good initial guess. So then the question is, can we find a good initial guess for this problem? And in this case, you can because you can use the explicit formulas to again provide the initial guess for this implicit formula. So that's what they do, also.

And so now, if we're doing an iterative procedure in order to get the initial guess for our other iterative procedure ever there, which we're going to break up into another iterative procedure inside it to solve every step. But this is currently what people do. So you guys are smart. Maybe you can figure a better way to do it. The world is waiting. I think I'll stop there.

By the way, actually just names-- This kind of thing is called method of lines. And if so, if anybody ever says I solved something by method of lines, what they meant was, they discretized in some of the coordinates and they kept one of the coordinates as a differential. And then they solved it with an ODE solver at the end. And in certain systems like this, it's a smart thing to do. You need to be kind of lucky that the set up is right, so you can use it. But if you can, it can be pretty good. All right. See you on Friday.