WILLIAM GREEN JR: All right, so we're going to start a new topic today about ordinary differential equations initial value problems. But before I start to talk about that, I want to remind you next week's schedule is weird. So there's no classes on Monday, but instead we'll have Monday class on Tuesday. So I'll see you next on Tuesday morning.

And then we're not going to have a class on Wednesday, but we have the quiz on Wednesday night. And then we'll get back to normal on Friday. And I would expect that possibly the TAs might be interested in giving a help session or review on Wednesday the class period.

AUDIENCE: We'll be here at 3:00.

WILLIAM GREEN JR: They'll be here. So if you want to come at class period, and just ask questions and stuff.

AUDIENCE: And I'll have your PSAT graded back, if you like.

WILLIAM GREEN JR: All right, got that? PSAT be ready as well. OK, so I'll talk about today ODE-IVPs.

AUDIENCE: Yeah, we'll still have office hours Monday [INAUDIBLE]

WILLIAM GREEN JR: Can you guys hear this? Yes, office hours like normal Monday. And for those of you who want to get your homework started early, we might even post homework. And I'm going to focus exclusively on ODE-IVPs you can write this way.

All right, and I would just comment, what do these things mean? ODE means ordinary differential equation. It means that the only differentials in the problem are with respect to one variable, so I'm going to call that t. It can be z or x or whatever you want, but I'm going to call it t here.

And so this is not partial differential equations where you have differentials respect to many

variables. Initial value problems mean that all the initial conditions are given in at a single t point where t is that the variable that appears in the differential. All right so this is the form we're going to do.

This is first order a way of writing the ODEs. If you recall from homework zero problem number one, we showed you there how you can write higher order differential equations-- convert them into the form of first order. And I guess that's worthwhile to point down. I would also comment that I going to just talk about f of y, however the ODE solvers in Matlab expect f of the t comma y as their from.

All right, so let's just explain all the details about this. So suppose I had a differential equation like this. All right, so this is like a differential equation you'd have for a body move moving-- a particle moving with some friction force on it and some time varying force. Yeah?

AUDIENCE: Are you going to be posting your notes online?

WILLIAM GREEN JR: No, I don't have any notes. This is it. All right, so this is a differential equation. What you can do is to convert it into the form I have above, is to write a new variable, say v, that's defined to be dx dt. That's the velocity. And has a time dependence, but I told you I don't like time dependence, so I'm not going to write it that way. So we can also define another variable.

So we can define our new y vector to be x, v, and t. So this is y-- the first component of y, this is the second component of y, this is the third component of y. And then I can write down what f of y is. So the equation for dx dt is y2. It's the second component of y, dx dt is equal to v.

The equation for the dv dt is given by this equation. So that's f of y3 over m minus gamma y2. And the equation dt dt is just one. So this is my f of y.

So this is f1, f2, f3. Is that all right? So this is just how you can convert-- if somebody writes you an equation this way, and you want to get it into the standard form that I'm going to show you here-- dy dt equals f of y-- this is how you convert.

And a skill we haven't really emphasized so far in the course, but I think a very important skill, is to be able to get like a chemical engineering problem and rearrange it so it becomes a standard form that maps up to one of the solvers you know how to use. So you've got a problem has some differential in it. Right away you actually, can I somehow rewrite it so it looks like this standard form-- or actually for Matlab, this form. So Matlab will say f of t, y.

If you can do that, then you can go ahead and use those Matlab solvers to solve it. But your variables may not have a letter y appearing anywhere, right? You have to figure out how to write it into the standard form. And then once you can do that, you can use the solvers.

And the same thing with the non-linear equations solvers, with the optimization software. If you can rewrite it into the form that matches the standard form, then once you're done, you're done. Then you can just go and call the canned programs that work well for those forms.

All right, now what do we want as the output? So this is the problem. This is the problem as proposed. What do we really want out of this?

What we're going to get out of it is a bunch of points y-- actually a matrix, right? It would be a vector of y-values. Typically y is a vector. And we want to know it a bunch of time points. And we would a lot of time points so we're getting pretty plots of how the components of y vary with time.

That's what we usually want. Sometimes we just want the final value, y time value. We integrate to some point and then stop, and just run through the final value. But a lot of times, we actually want to make pretty plots, so we really want a lot of points here and making plots.

Now how many points do you need to make a nice plot? Maybe, I don't know, 100 points, 50 points, you can make a nice plot. So that's the kind numbers you'd like. And so the output of these programs is going to be something like a vector of the time points, and then a matrix of the y-values corresponding to each time point.

And the way it does in that lab, it's reasonable. Each row of y are the y-values that correspond to the same time point, the first time point. So the first row of y corresponds to the first number in the time vector. The second run is the second. Last one will be at time final. And then you can plot them. OK so far?

And ideally, we would like these y-values that we compute, these y's we get out, ideally one that really close to what the true solution of the true differential equation is. But we're doing numerical stuff, so it's never going to be exactly the same. And so a big part of the effort is trying to figure out, how can we get the thing to compute y calculated at each time point to be as close as possible to the truth, of what the true solution of the equations are? And that turns out to be difficult, so that's a very important thing to worry about.

What else can we say about this? We specify this from t naught, and you typically have to

input your t final, how far you want to go away from your t naught as another input. Normally, people write it where t naught is less than t final, and so you're like integrating from left to right. But you could actually write them, put a negative sign equation which corresponds to a negative t, change your variable to of t prime that was equal to t final minus t if you wanted to, and integrate the other direction.

And in fact in homework zero problem one, we did that too. So you can integrate frontwards. You can integrate backwards. If somebody tells you the initial condition or condition in the center of the range, you integrate forward for part of the range, and backwards the other way, so anything like this.

So what's important for this to be what's called an initial value problem is just that all of the specifications of y are given at the same value of t. It doesn't matter exactly what value of t it is. Later, we'll talk about what happens if you don't know all the specifications at one value of time.

All right, now how would you approach this? Probably, a bunch of you've done this already as undergraduates, or maybe even high school. You can write like a Taylor expansion, y of t plus delta t-- is y of t plus delta t-- times dy dt.

You can write that. And then I say, wow, I know dy dt. That's actually f, so this is actually equal to-- right? And then if I truncate the Taylor expansion, now I have an update formula.

So if I just forget this last bit, I can put in my current value of y, like the initial condition. The initial condition here, evaluate f, multiply by delta t, add them together, and I'll get a new value of y.

And then I can repeat over and over again. And in fact, this is a very famous method. It's called the Forward Euler method. Euler was a pretty smart guy, so it's not completely ridiculous, but as I'll show you, it has some problems.

So the Forward Euler, another way to write it is y new is equal to y old plus delta t times f of y old. It's called Forward Euler. And we can write a little implementation of this pretty easily. So you can write y is equal to y naught, t is equal to t naught for i equals 1 to the number of steps, y is equal to y plus delta t times f of y, t is equal to t plus delta t. Somewhere up here, I need to write delta t.

OK, so there's your code for doing Forward Euler. How many of you have done this before? OK, so I can skip over this very fast. How many did this, and it didn't work for a problem? OK, so you're not doing hard enough problems. Well, we'll correct that.

All right, so I noticed that this closely resembles the rectangle rule. So the rectangle rule would look a lot the same. However, if you're integrating a function-- so I have I is equal to the integral of f of t dt, then I can notice that the derivative dI dt is equal to f of t.

And so if I was going to do the rectangle rule, it would look just the same as this, except this would be a t. Maybe I would pick y0 to be 0. And the final value of y I would get to would be my integral, my i value that I want. So how many of you have done rectangle rule? Yes? All right, so you did this already.

Now you remember in high school, when they took the rectangle rule, they probably mentioned that it's not very accurate. And so then they also taught you some other ones. You guys, how many of you learned the trapezoid rule?

How about the midpoint rule? How about Simpson's rule? All right, so at least a high school math teachers thought that this is inadequate for doing real work, so they tell you all these other things instead. This was like your first baby thing.

So let's talk about why was that. So let's do the Taylor expansion a little bit further. So let's do it for the numeric [INAUDIBLE] case. So y t plus delta t is equal to y of t plus delta t times f of t plus 1/2 delta t squared times the second derivative.

And so when we made the approximation to do the rectangle rule, we just threw this term away. So that's a leading term that we threw away. So we made an error order or delta t squared. Because normally the second derivative is not exactly zero.

So we have an error in each step that's the order of delta t squared. But how many steps do we make? The number of steps is equal to t final minus t naught over delta t. So therefore, we're making one over delta t order steps.

So the total error is going to be approximately the number of steps times how much error you making each step. So it's going to be something the second power in delta t divided by something to the first power of delta t. So the total error in the integral I is going to be older of delta t if you do the rectangle rule.

And so order delta t is not very good. So you want to increase your accuracy-- if you cut your step size in half, that means you double your CPU time, because you have to do twice as many function evaluations. But you only gain-- you reduce your error by a factor of two. So if you want to, say, gain three significant figures of accuracy in your number, then you have to do 1,000 times much work as you did before.

So I have some amount of some precision with some number of steps. I want to increase, get three more significant figures. I'll have to do 1,000 times as much work. That's kind of bad scaling. If I want to get six more significant figures, I've got to do a million times more work. So at some point, this going to be kind of expensive.

So if you contrast it, you can do things like trapezoid rule. And as you're probably-- well let's talk about trapezoid rule for a bit. So the idea of this is you have your function, you have some point t, and t plus delta t. You're trying to integrate between them.

Let's say t naught, t naught plus delta t. Try to integrate. Here's the value of the function at t. Here is the value of the function that t plus delta t.

If you do the rectangle rule, what you say is I just draw a line here and integrate that rectangle. But say the real function really looks like this. Then I missing that area there, so that's the error.

That's why the error's so big. If I do that trapezoid rule, then the error-- I actually overestimate the value here, but the integral of the difference between this dotted line and the solid line is less than the integral between the solid line and that dotted line, so its more accurate. So that's why people like the trapezoid rule better than the rectangle rule.

If you did the midpoint rule, you'd evaluate this function halfway between these two points and then draw a dotted line here and a dotted line there, and integrate that guy. And it would overestimate for part, and underestimate for part, and they would partially cancel each other out. And so that also would be more accurate than doing the rectangle rule.

So you did it before. And so what we're going to try to do is, instead of using this formula we did before, we're going to replace this with something else that I'm going to call g. And g is something that's supposed to be the average value of f over the step. Instead of just taking the value f at the starting point, I want to get sort of the average. Right, because an integral is related to an average over the delta t.

So g is going to be my way to estimate the average. And there's going to be a zillion update formulas that different mathematicians have proposed over the years. They give you different g's that tell you how to do the update. And the key is try to find a g that's really accurate.

You want to have one that's the most accurate you can, because then your errors will be less. And if your errors are less, than your delta t can be bigger, which means your number of steps will be less, which means your CPU time will be less. So you might be able to get more accuracy and use less CPU time both if you can find a really good formula for g, for the update.

So what's the g for the trapezoid rule? It's f of t plus f of t plus delta t over 2. That was the g you used when you did the trapezoid rule when you were kids. What is the g for the midpoint rule? It's f of t plus delta t over 2. And so the different g's are called different update formulas.

And these particular ones reduce the error from delta t squared for [INAUDIBLE] step to, I think, delta t to the cubed power. And then when you multiply by this 1 every delta t, it turns out you go from being order of delta t to order of delta t squared, and that's a really big change. So now if I cut the step size by a factor of 10, I gain a factor of 100 in the accuracy. If I want to get six more significant figures, I do 1,000 times more work, not a million times more work. So it's a pretty significant improvement.

And then people have pushed this on further and further. And so actually very common integrators that you might use nowadays would go out to fourth and fifth order methods. And so they have complicated update formulas that are carefully designed to cancel out all the low order delta t errors and just leave very high order ones.

And that's kind of the-- typical ones are like that. And then some fancy ones might go even up to eighth order, I've seen, if you're really pushing it. You have really complicated g formulas then.

Now, this is all for the integration, but really our problem was that the ODEs [INAUDIBLE]. So we have to do f of y not f of t. Let's see if I can show you that.

Yeah, so we did the rectangle rule, we had f of t. But the real problem is we have to do f of y. Now the problem is, we don't know what y is, y is our unknown. So we generally have a problem here.

In Forward Euler, we can get away with it because we just put the y old value in. Like for example, if we wanted to do midpoint here, we'd have to know f at a different time point that we haven't computed yet, because it's forward in time. Same thing with trapezoidal rule.

We have to know f at some future timepoint. So it's not so easy, actually, to go directly from these formulas to some explicit formula like this with a g in here, where the things inside g, the y's inside g, are all y-values we actually know. So this is a serious problem.

But this didn't stop people from doing it. So one idea is you could go back to the Taylor expansion, and now do the Taylor expansion but in terms of f of y. So this is f of y of t. And then over here, this is d squared y dt squared, but dy dt is f, so this is really df dt. But f is actually not a function of t, it's a function of y, right?

But we can do chain rule. So df dt is equal to the sum df dyn dyn dt. But dyn dt is f. And this is what we call the Jacobian [INAUDIBLE]. So this is really saying that this is equal to J times f.

So that would be one possibility, is you could go and write the Taylor expansion out to the next higher order explicitly, putting in J times f here, and then just being in the cubic terms. Now, people don't do this usually. Any idea why this is not a popular thing to do? Yeah?

AUDIENCE: It's computationally expensive to [INAUDIBLE].

WILLIAM GREEN JR: That's right. That's right. So how many function evaluations does it take to get the Jacobian? Right, so each function is n values, right, because it's a vector. And you have to do at least, say, forward differences would be n function evaluations. So instead of just doing one function evaluation like we were doing with the Forward Euler, now we have to do n plus 1, or something like that, function evaluations.

And actually, you probably want to use center differences, so you need 2 times n plus 1. And if you do them analytically, unless it's sparse, it's still going to be really expensive to compute it. So it's a lot more effort to do it.

So you need to think, well, is it really worth it? And so people have found other formulas that basically get rid of the delta t squared term that don't require so much effort, and so that's what people do instead. But this is an option. You could do it.

All right, so instead, what people do is they've decided to generalize the midpoint rule and try to figure out, how can we get an estimate of the midpoint value that's cheap to evaluate? And

so what they say is, well, this is g midpoint for numerical integration. Let's make an new g midpoint that works for when f is a function of y. So let's make it function of y evaluated at t plus delta t over 2.

So that's what the formula looks like. But then you say, well, I don't know y is at t plus delta t over 2, because I only know it at y of t. So then they say, well, let's do Forward Euler for that. So then they say, well, let's say it's approximately f of-- what's the Forward Euler formula?

It's y of t plus delta t over 2 times f of y of t. How many parenthesis there? So this is the formula that's actually used in practice a lot. And this one is called Runge-Kutta two, second order Runge-Kutta formula. And it's just a-- it's the midpoint rule where we estimate the value of y of t plus delta t using Forward Euler.

And the advantage of this is it's very cheap, right? Just evaluate-- I just have to evaluate one function here, one function here, so two function calls. [INAUDIBLE] and that way I can get an update formula. And this one, turns out that it has the nice delta t scaling that you might like.

So that's the kind of thing people do. And the Runge-Kutta guys went crazy, and so one of the most popular solvers is called Runge-Kutta four five, and that's the ODE four five program in Matlab. And what that does is the fourth order extension of this, and the fifth order extension of this.

And it uses them both, and it compares them, and uses the difference between an estimate the error in the calculation. And then uses that to decide what size delta t you need, depending on what tolerances you demand. And the formulas for all those-- [INAUDIBLE] formulas are given in the textbook.

Now, this starts to lead the problem. So we weren't able to actually evaluate the true g midpoint, which we'd like to be here. Instead, we have to use some approximation in order to extrapolate forwards to the y. So this is hinting at the whole problem.

This whole problem is actually we're just extrapolating. We know the true value of y only at t naught, and all the rest we're doing is extrapolating forwards. And we're do it over and over again, so we're going to do it for n steps. Maybe we'll do 1,000 steps.

So you're extrapolating. And then based on that extrapolation, you going to extrapolate again. And then based on extrapolation, you going to extrapolate again. And you can see that this is not really the most robust thing to do, right?

Because if you have any experience, you're not that comfortable extrapolating at all. And then you have to extrapolate 1,000 times. You should be a little bit worried about what can happen.

So let's think about how the error will propagate. So we're going to have some errors while we do these extrapolations. So we have here a t naught, t naught, we're happy. We know y naught. This is exact. We're like woohoo. We know one value of y, really good.

All right, so now the question is the first step. Now we're going to get to value y1 that we compute with one of the formulas, and it's going to be good to some accuracy depending on how good or g is, our update formula. And so it's going to have an error.

I should comment that this y naught, although we treat it as exact, it could have an error too. But it will probably be more like a machine precision kind of error because we don't really know the initial conditions that well. Or might be an experimental error about how well we know what the initial conditions are.

But I'm not going to worry about that one for now. But be aware, this is true, we don't really know initial conditions perfectly either. But certainly, even if we treat this as being known perfectly, we're going to have some error by the time we extrapolate to y1.

So now what happens at y2? So now we've added some more, so now we're at t2. And we see started from y1 true plus some error. This is y-- y naught was true. Now we're starting from y1 true with some error in it, because of a mistake we made the first step. And now we see how that step propagates further.

So let's see what that's going to say. Well say y2 is going to equal y1 plus delta t times some update formula which would depend on y1. But y1 was not true anymore, so this is actually equal to what should have been the true value of y1 if we only knew what it was, but we don't know. And we have our error that we don't know exactly how big it is that we added on. And then we have another term from this guy, so it's going to be delta t times g of y1 true plus delta 1.

And then we know our update formula's not right anyway, so there's actually another error, delta 2, just because of the error in the update formula. Now if we had a great update formula, these deltas are kind of small. And if somehow, by luck, our delta 1 was 0, and we had a great update formula, then we think that this error in y2 is going to be pretty small.

But it's not true. So really the errors are pretty big. So let's see if we can figure out-- we're going to do a Taylor expansion here, so this'll be g of y1 true plus d dgy-- the Jacobian of g with respect to y-- times the delta 1. That's the first order Taylor extension.

All right, so now let's group the terms. So y2 is equal to y1 true plus the update formula of y1 true plus the error in our update formula if we had started at the true value. So this is what we-- if somebody told us what the true value of y1, this is the size error we'd expect. It's to be that little error we calculated before.

But then on top of it, we have these other terms. We have a delta 1, because y1 was not the true value. Sorry, I lost a delta t here. And then we have delta t times dg dy times delta 1. And I'll emphasize these are all vectors. I think that's it if we only keep to first order.

So what is this thing? This is a matrix, right, dg dy. I could write it this way. It's the identity matrix plus delta t times the Jacobian dg dy times the vector d1.

And you can see, at every iteration when I do this, I'm going to get a similar factor like this. And it's going to again multiply delta 1. I keep on multiplying the-- errors from my previous step keep getting multiplied by this kind of factor.

So it's going to be really important to us about what the norm of this matrix is. What's going to happen if the normal of this matrix is big? What's going to happen?

AUDIENCE: [INAUDIBLE]

WILLIAM GREEN JR: That's right, the error's going to get multiplied by some factor, say bigger than 1, every iteration. And after we go 1,000 iterations, how big is the error going to be? Really big, right? Even if that thing is quite close to 1, 1 plus something to the 1,000 power it's gigantic.

So what was originally a pretty small error-- because we chose a good g method-- is going to get multiplied by this huge amplification factor. So the key thing for a method to be what is called numerically stable is that the norm of this matrix has got to be less than 1. Or maybe even equal, equals 1 might be OK. If this is much bigger than 1, you're doomed. Your numeric order is just going to blow up from step to step to step.

And what's bad about it, is actually sometimes you might not even be aware of this. So you'll get some solution, because it still will calculate some numbers for the y's at each type-- y1, y2, y3, y4-- it's just running through the calculation. But the numbers may become increasingly

meaningless further away from the y true as time goes on, as the steps go.

AUDIENCE: [INAUDIBLE]

WILLIAM GREEN JR: Yes, it can. Yeah, so that's what we call a stable numerical method is one that, if you make an error in an earlier step, its impact diminishes as the steps go on. That's what you want. You can't always achieve this.

But if you can, that's really great. Then that's you would call a really stable, robust kind of method. Even though I give you some stupidity early on, my stupidity evaporates away, and it goes to the true solution. That's what you want.

OK, so let's think of cases where it's going to be problematic for sure. So if dg dy-- say if all it's eigenvalues are positive and you add it to identity matrix, the thing is still going to have eigenvalues bigger than 1. Can you see that? And actually, what do we say when dg dy is positive, has a positive eigenvalue? You guys have a word for this already you learn in your differential equations class. What do you call these kind of differential equations?

Maybe not dg dy. Back up, how about df dy? If df dy, if that Jacobian has a positive eigenvalue, what do you say? You guys remember this? So this is what we call unstable differential equations.

So if it has any positive eigenvalues, then two initial conditions that differ by a little differential exponentially separate as time goes on as long as the difference as any projection in the direction of the bad eigenvector. So those are called numerically unstable. A lot of those actually exist in reality.

So explosions, that's because you had some positive eigenvalue somewhere for example. Amplifiers, like you buy an amplifier to crank up the music, you get positive feedback when-- remember Professor Swan walked to some strange place, and for some reason, he got the feedback from the speakers? So he had a positive eigenvalue going there. A little tiny noise and his microphone, and it amplified to a big squeak.

So there are real systems like this that amplify. And sometimes we want to model them, like explosions for examples. That's a pretty important for chemical engineers.

But it's going to be really tough, because if it's got positive eigenvalues, then this is probably going to have a norm bigger than 1. And then whatever little errors we may get any step in the

whole calculation, we're going to amplify and amplify and amplify. And who knows if we're going to have any reality left in our solution by the time we get to the end.

All right, so that's one kind of problem. So how about we have a problem that's intrinsically stable. That means that all the eigenvalues of the Jacobian of the original problem are negative. So the thing should be perfectly stable.

From whatever initial condition they started from, the whole thing should sort of come down to like an equilibrium point. So that's a pretty common situation in chemical engineering too. You start from some state, and it relaxes down to like an equilibrium state or a steady state. Well-behaved systems act like that.

So what I'm going to show you is that, despite the fact that the df dy is well-behaved, it's possible that this sum matrix might still be poorly behaved and still have a norm bigger that one. And so that means you started for a problem that was pretty well-behaved, and you broke it by your choice in numerical method. So this is a very bad one. This is the kind that can make you really embarrassed if you're working in a company.

Your boss gives you a problem. He says, please tell me the time is going to take our reactor to relax after a start up. the system is perfectly well-behaved. He gives these beautiful equations that your previous coworker worked out. Everything's fine.

You put it in there. You use your stupid numerical solver, and you get a wild oscillation, and the thing blows up. OK, and that's going to be because of a poor choice of g so that it makes this norm of this matrix bigger than one. It's pretty easy for that to happen. So you don't need a very complicated case to show it.

So let's just do a scalar case. Say dy dt is equal to negative 2y. So this is a well-behaved differential equation. What's the solution? So with a y naught y at t equals 0 is equal to 1. What's the solution?

AUDIENCE: [INAUDIBLE]

WILLIAM GREEN JR: Yeah, OK, so it's perfectly well-behaved function. And the plot when you plot it, here's one. It goes [PLANE DIVING NOISE]. So now let's solve this with the Forward Euler method.

So I'm going to say that y new is equal to y plus delta t times f. So in this case, this is f, right there, negative 2y. So for Forward Euler, this is just saying that this is negative 2y. All right, so

let's choose a delta t, say delta t equals 1 to start with.

So we're going to compute. We start at t equals 0. We're computing out to t equals 1. so we're out here. So we put 1 in here. Our initial condition was y was 1, so that's 1. So as 1 times negative 2 times 1 is negative 2. This is a 1. 1 minus 2 is negative 1.

So at the first step, it goes from here down to there. This is not looking good. This physical variable is supposed to be gradually relaxing towards 0. Going negative is not what you want. And then I think we could go the next step, if you want.

So we can put in suppose y is negative 1. And we're still doing a delta t of one step. So negative 1 times negative 2 is positive 2. Plus negative 1, so this goes back to the starting point. We'll get the sawtooth, is what you get.

And it turns out, actually, if you make delta t any larger than 1, then this sawtooth amplifies, so you have a oscillating solution that explodes. Which is a little bit different than the physical solution, which gently relaxes. And so this is just a scalar equation that you guys could have done in high school. If you high school math teacher had been brave, he would have showed it you. But he don't want to ruin your faith in the Forward Euler method, so he just showed you a simple one that worked out.

Now you can make this work better by making the delta t smaller. If you made the delta t small enough-- in this case, I think it's delta t is less than half-- then this actually behaves somewhat reasonably. And so what's going on there is that the identity matrix is positive. My Jacobian is negative. And if I make delta t small enough, this big positive number is bigger than this smaller negative number, and the whole thing stays positive but less than 1, and so then the normal is good.

But if I make delta t too large, I make this second term much larger. If it gets much larger than 1, it overwhelms the first term. The thing's negative, but I what do the norm it's positive. And so it has an expansion. Then the negative is why we get these oscillations.

So if you ever do a numerical differential equation solution and you start seeing this kind of stuff, it's almost certainly related to this numerical instability of the method. and not really a physical thing. I mean, there are physical things that do this, but not too often in chemical engineering, fortunately. All right, questions? No questions, OK.

So in the textbook, they have a big discussion about how to evaluate different g methods in

order to figure out if they're guaranteed to be stable, or under what conditions they'll be stable. And so that's worth a look to see. And they have detailed derivations for several of them.

Questions? All right, so for the homework, homework four, you're going to have to write your own ODE solver. So you'll do update formulas and compare them. When we come back on Tuesday, I'll show you how this is usually dealt with by switching to implicit solvers and talk about that. All right, have a good weekend. Enjoy the three days off.