[SQUEAKING]

[RUSTLING]

[CLICKING]

FRANK SCHAFER: Yeah, so, yeah, I plan to split it in two parts, so the first part is about ordinary differential equations, so just a quick recap what an ordinary differential equation actually is, and then we'll go into this kind of sensitivity analysis, so how to compute derivatives of solutions of ordinary differential equations. And specifically, we will pick out one very important of these methods. So, yeah, an ordinary differential equation is basically described by this kind of initial value problem, where we have the state at a given initial time t0 is equal to u0, and then the dynamics of this state is described by this kind of time evolution.

So we have that the derivative of u is-- the time derivative of u is equal to this function f that can depend on time, on the state itself, and on some parameters p. And the first natural question to ask is, basically, how do we solve such an ODE? And let's say the probably simplest ODE I could imagine basically is one of a free-falling particle or a ball, which is described by-- well, probably, you have all seen this-- by Newton's law, or something like the acceleration is equal to minus the gravitational constant.

And for these kind of easy ordinary differential equations, right, we can solve that analytically, so we simply integrate that equation twice, right? So we have that z prime is equal to some offset minus g t minus t0. And then we could integrate it a second time, and we'd get, like, the full analytical solution of the height, in that case, with respect to time.

Here, on this slide, I did already something that's very meaningful to solve ODEs numerically, and that's, like, we want to bring down these kind of ODEs, which are sometimes described with second-order derivatives, to just first-order derivatives. And the basic trick is to introduce this additional variable v, right, so the velocity in that case, which can be written as-- right, so v of t is equal to dz of t dt.

OK, so for that kind of ODE, we know the analytical solution, and it's basically easy to solve. Now, coming back to the more general case, where we have some nice function f, but solving for the analytical solution is tough, right, because integrating is already tough and solving these kind of initial value problems is even tougher, we can fall back to some numerical routines, basically, to solve it. And the easiest numerical routine is called Euler solver, where we-- and probably all of us could have come up with that by ourselves-- so we simply discretize time.

So we introduce this tn, discrete time steps of our solution, and then if we plug that into the definition of the derivative, we get something like these finite difference equations. So, right, if we see here, the du dt would be u n plus 1 at the-- so that's basically the state at the time tn plus 1 minus the state at the time tn divided by delta t, and if we bring the delta t and the state to the other side, we get this explicit Euler method.

STEVEN JOHNSON: And last week, we talked about finite difference approximations, so this is exactly taking that derivative and making a forward difference approximation.

FRANK SCHAFER: Yeah. Yeah, exactly. And you can have more sophisticated methods, of course, to solve your ODEs, but for the rest of today, it's basically enough to just think of, if we don't know the analytical solution, we can use some numerical routine to solve for the state as a function of time. And, yeah, actually, I wanted to implement that very quickly with you, like, for the case of the free-falling ball, so for that kind of ODE.

So let's define some initial height, some kind of initial velocity, the gravitational constant, and, well, maybe let's say that's basically our parameter, g. We can say our initial state, right, is then this vector where we stack together initial height and initial velocity. And we say maybe, in our task, we want to integrate that equation from some initial time to some final time, and we have to provide this kind of time discretization. Let's choose some small time discretization.

Everybody's happy with that so far? So then we want to define our ODE function, which is also called the vector field, where we just insert the definitions I had on this slide, basically. So we have that the derivative of the height with respect to time is the velocity, right, which is here, in our case, now the second component. And we have that the derivative of the time is equal to-- the derivative of the velocity with respect to time is equal to minus g, which is on the right-hand side of the board, and then we return, basically, again, the vector of these two.

STEVEN JOHNSON: We're going to get g from the parameters, p [INAUDIBLE].

FRANK SCHAFER: Oh, yeah. Let's say it like that. Thanks.

And so now we want to implement our Euler solver, which should take-- well, the kind of function that we want to solve, we want it to take the initial condition. Let's write it a bit more general. The time span of the simulation--

STEVEN JOHNSON: tspan.

FRANK SCHAFER: tspan, yeah, and the parameters, and maybe even dt. And let's say our time at some point will be-- so we make a copy so we don't overwrite into the array. So that this is preserved, we make a time grid. This is actually big enough on the screen for everyone?

Oh, good. Right, we have to be a bit careful with the endpoint because we don't want it to step one too far. And then the update rule is basically we take u at a given time, and that's-- whoops.

STEVEN JOHNSON: And you don't need the deepcopy since you're not overwriting u in place anyway.

FRANK SCHAFER: That's true. I think I will need it later for one modification, so--

STEVEN JOHNSON: But you don't need any copy.

FRANK SCHAFER: That's also true. OK. And let's say I'm not interested in the full solution as a function of time, but just in the endpoint as-- OK, let's see if that works already. All right, that's good. u should be at ti, for sure.

Yeah, we can check if that's the correct solution by plugging it into our analytical equation. Let me just pull the form. Right, so we have now written a numerical routine that spits out the state at a given time t, and I think that looks actually good. So we have t equal to 0, so our z should be z0 minus g-- minus v0 times 1 plus v0 times 1 minus 1/2 times g. And, yeah, that looks basically good. That's the discretization error of the Euler method, and we could check the same for the second component.

STEVEN JOHNSON: Yeah, so the correct answer is 0.1, and then it's-- it's, like, 0.105.

FRANK SCHAFER: Yeah, it's-- yeah, exactly, it's slightly off because we have not the most sophisticated numerical solver. OK, cool. So that's how we solve ODEs numerically if we don't have an analytical solution, and, yeah, now we get into the kind of sensitivity analysis if we have a numerical routine available.

So there are two different ways, fundamentally different ways, to do sensitivity analysis. One is called discrete sensitivity analysis, and one is called continuous sensitivity analysis. And the first one is basically you imagine that you wrote that solver that we just wrote in some language that's compatible with automatic differentiation. And so we first discretize, basically is the task, and then we differentiate it.

And the second one is basically we write a custom rule for the sensitivity analysis, and so we first approximate, basically, the gradient that we want to compute and then differentiate it-- and then discretize it. And for each of these methods, these different paradigms, basically, in either first discretizing and then differentiating or first differentiating and then discretizing, there are two different modes that you also probably have covered in the lecture already. You can either derive a forward, or also called tangent, mode or a reverse, adjoint, mode.

And depending on if you have many input parameters and a small number of outputs, one of these modes is basically preferred to each other. So in the setting where you have many parameters with respect to which you want to compute a gradient or derivative, you usually want to have some adjoint mode, and if you have high memory demands, you typically want to have these continuous sensitivity methods. So it's a very relevant case to consider these continuous adjoints, basically, and that's what we will do today.

So what are sensitivities good for? Just as a recap, so sensitivity analysis basically just, well, describes what it says. So we might be interested in just describing how maybe the height of our ball would change if we slightly perturb the parameter of the differential equation, but we can also use the gradient of these ODEs to do more fancy stuff, as Steven mentioned, like parameter estimation in, what, chemical reaction networks or so, or control for whatever physical system or chemical system you have at hand.

So once again, so for the free-falling particle, it might be the task that we have some terminal cost or so, that could be this G of u at the final time, and we are interested in maybe saying how the height changes with respect to the gravitational constant. And so the claim is that this is always easy if we can solve the ODE analytically, right, so if we look at the solution there of the ODE, can someone say what's the sensitivity that I wrote down, like, the sensitivity of the final height with respect to g? Yup?

AUDIENCE: Negative 1/2 t minus [INAUDIBLE].

FRANK SCHAFER: Yeah, exactly, so and now we will basically cover how we do that if we don't have the analytic solution available. For this, we will have to do some derivation. Uh-huh. Yup.

Let me move them. So we'll do continuous adjoins today. So let's first write down one of these cost functions which I just had in a bit more general way.

So we have some cost that depends on the state as a function of time and p, where the state itself, of course, also depends on the parameters. And we'll say that this function can be written as the-- all right, so we say we can-- we don't have, maybe, just the final value, but we are interested in general in some functional on the solution of the differential equation.

AUDIENCE: And you can think of capital G and little g as just being scalar functions?

FRANK SCHAFER: Yes.

AUDIENCE: OK.

FRANK SCHAFER: Yup. Yup. And now comes basically the first nice trick in this kind of derivation, and that's related to Lagrange multipliers.

STEVEN JOHNSON: Do you want to write down the ODE that at least satisfies?

FRANK SCHAFER: Yeah, probably a good idea. All right, such that-- Oh, I see. OK, so the first trick that we want to do is basically to add a 0 to this kind of function, so--

STEVEN JOHNSON: Can we pause to make sure we're understanding what you're doing? So you have some ODE, you solve for u of t, and then you're going to compute a number at the end, capital G, by keeping the ODE solution and integrating some function little g into that solution, right? And the ODE depends on some parameters, and you want to find the derivative of this final output, capital G, with respect to all those parameters of p.

So little p might be a vector, it might be 10,000 reaction rates. And capital G might be the integral of the squared difference between the ODE solution and some experimental data on the chemical reaction, you know, the concentration, [INAUDIBLE].

FRANK SCHAFER: Yeah, exactly. Does everybody see why that's a 0? Right, so why--

STEVEN JOHNSON: Yeah, so, sorry, I was talking about that and then maybe you need to say what the 0 is

FRANK SCHAFER: Right, so because, by definition, basically, our state fulfills this ordinary differential equation, we can introduce this kind of Lagrange multiplier, and that term is 0. So if we add that term to this kind of cost function, we basically do nothing, but we have this free Lagrange multiplier that we can choose later on. So that will be kind of the trick that we will play.

And now, as Steven already said, so we're interested in computing dG over dp, right, so that's the kind of sensitivity gradient information that we want to have. There will be a long line, so what we basically do is just doing all the chain rule on that term and these terms.

STEVEN JOHNSON: And now think of p as a scalar, but in general, it could be a vector or something. And if it's the gradient, of dG dp.

FRANK SCHAFER: Yeah, it's a gradient transpose, yeah.

STEVEN JOHNSON: So what we were calling G prime in the book.

FRANK SCHAFER: Yeah, let me quickly write it down because it's easy to do mistakes on that part. Oh, minus sign from the minus we had before.

STEVEN JOHNSON: Maybe you could write it a little bigger?

FRANK SCHAFER: Oh. OK. OK, I'll leave out the utp's to save some space, but every u here depends on t and p. I guess that's maybe hard to read at the end, but there's a dt only missing here. Is that readable for you on that side?

AUDIENCE: [INAUDIBLE].

FRANK SCHAFER: I know. I will repeat that term in a second, so you will see again what's that term. Right, so let's call that basically, like, the sensitivity, call that, and yet, let's simplify a bit the notation.

So that's my small g. I'm sorry, my small g's and small f's look fairly similar. That's small gu, so I'll always denote by subscript with respect to what we differentiate.

STEVEN JOHNSON: This is like in the homework, you know, has a few problems where partial derivatives are subscripts.

FRANK SCHAFER: Small gp, and in this term, what we can do here is basically swap the derivative with respect to p and with respect to time. So we get here a prime, right, so because when we swap it, we have du dp, which is s, and then we have the time derivative. And we have here some fu times s, and the last term is f-- that last term is fp. OK, let me go to the center because I think it's probably better readable from there.

STEVEN JOHNSON: Can you scroll the board down so we can see more?

FRANK SCHAFER: Oh, Yeah.

STEVEN JOHNSON: All right.

FRANK SCHAFER: Oh, OK.

STEVEN JOHNSON: Thank you.

FRANK SCHAFER: Oh, yeah. OK. So, yeah, I wrote this second term, which was probably hard to read, a bit bigger. So that's the second term on the right-hand side and--

STEVEN JOHNSON: Could I maybe say something to try and clarify?

FRANK SCHAFER: Yup.

STEVEN JOHNSON: So, right, so you have this function you're trying to differentiate, and if you just differentiate this p directly, you get this term, which is easy. Right, if the function depends directly on the parameters, you can take that derivative. Often, that derivative is 0. This term, which is easy, this is your cost function, how does it depend on the solution.

This term is hard. How do you get the derivative of the solution with respect to the parameters? So the goal and adding 0 here is going to be-- is going to give us this extra degree of freedom, lambda, that will-- if he's clever in how he chooses this, there'll be a trick that allows him to cancel this term this term that's hard and get a differential equation for lambda. So that's-- it's often good to sort of show-- there's kind of a magic step here where you added this lambda term, and you know, sometimes it's good to not keep people in suspense as to why the hell did you add this term.

FRANK SCHAFER: That's true. Yeah, exactly. Yes. One step before we can actually do that trick to eliminate the s, and that's basically this s prime that we got, which we first have to get rid of. Anybody has an idea, maybe, how to do that, like, our plans to turn s prime into an s?

STEVEN JOHNSON: 18.01 question, you have a derivative in an integral you don't like, how can you get rid of it?

AUDIENCE: Integrate by parts.

FRANK SCHAFER: Yeah, so we integrate that term by parts, and that gives us-- all right, so that's from the integration by parts, now the prime on the lambda, which will be exactly going towards the ODE that Steven mentioned.

STEVEN JOHNSON: Time is always a time derivative?

FRANK SCHAFER: Yes. Yes, I should have said that.

STEVEN JOHNSON: OK, can you also comment on the shape of lambda? Like, why is lambda-- is lambda a matrix, a vector, a scalar, what is the-- why is it transpose? Like, what's the size of this lambda?

FRANK SCHAFER: The size of this lambda, so it's such that-- right, this output should be a scalar, right, because we optimize the scalar cost function, so the size of lambda is basically in the-- well, in the adjoint space of f, right-- or of u prime.

STEVEN JOHNSON: Yeah, so the solution here is not a scalar. u is not a scalar. It's a column vector, right? You have n equations and n variables. Lambda is a column vector with the same number of components and has the same-- it has the same shape as u.

FRANK SCHAFER: Yeah, so what we want to do next is basically insert back this term into that equation, so we-- that was our term here. And so we take now this part, insert it back, and we're already going to what's our goal, and reorder all terms such that we have this s separated, basically. So what we end up with is basically this dG dp.

STEVEN JOHNSON: It's a little funny because in Greek-- in linear algebra, we're used to Greek letters being scalars, but for adjoint methods, the adjoint vector is always called lambda because it comes from Lagrange multipliers.

FRANK SCHAFER: Oh, I'm missing a prime here. OK, let's write that directly into one. OK, so not much happened here, basically. It's just looking at all the terms. So we leave that term here. This is that term. That is that term. We moved this term which doesn't depend on f to the front, and we had another term that was depending on S here, which we moved to the back, basically, so these two terms were, like, swapped.

AUDIENCE: So again the goal here is-- the S, which is d u-- the derivative of the solution's expected parameter, that's the annoying term you don't know how to deal with, so you're going to try and group it all together and then make that term 0 so that you don't have to deal with it.

FRANK SCHAFER: Yeah, so we are free to choose lambda, so we can choose lambda to be whatever we want. So why don't we take basically the choice lambda prime of T is equal to minus lambda T fu minus gu? Right, if we take that choice, we will have that this term vanishes, and we can also, since this is an ODE, specify, well, basically the initial condition.

So we can specify that lambda of t should be equal to 0. If we do that, we have that this term cancels, and one of the terms from this integration by parts also cancels. Then it's the question of what--

STEVEN JOHNSON: And why did you choose capital T to be that, lambda capital T to be 0?

FRANK SCHAFER: Yeah, that's the next question, and that's because-- basically, what is s of t0, which comes into that term, so we know that's s of t0, right, by definition of s, it's the sensitivity of the initial state with respect to the parameter. And, right, and we know that the initial state doesn't depend on the parameter, so that term--

STEVEN JOHNSON: Or it could, right?

FRANK SCHAFER: Oh, it could.

STEVEN JOHNSON: It could, but whatever it is, you know what it is. Like, you don't-- you have the initial condition in closed form, right, so even if it depends on some parameters, you know what it is. But here, here only the-- you're simplifying it to-- only the system of equations depends on the parameters.

FRANK SCHAFER: Yes. Yup. Exactly, so either it's easy, or it's, well, 0, basically.

Yes, nice, so we have in the end that just this term, basically, survives, so we have that dg, the derivative of g with respect to the parameters-- or the gradient, in this case-- is equal to that derivative, which is easy to compute. It depends on this lambda, and it depends on the derivative of f with respect to p. That's a Jacobian, basically.

So let me let this here. Right here, so what's kind of the procedure that we need to do to solve it, right? So when we solve that integral, we realize that we first have to solve this ordinary differential equation, which gets a special name, actually. So this is the adjoint ODE.

But we also see that the forward solution is embedded in that ODE, right? So to solve for the quantity which we're interested in, we basically have a three-step procedure. So we first solve for u of t. Well, that's at least the easiest way to think about it. We first would solve for the ODE for u of t, then we have that information. In the next step, we solve for lambda of T. I can also--

STEVEN JOHNSON: Can you write down ODEs again and just these steps? Just to summarize, you're solving an ODE for u of t, and write down the ODE for lambda, maybe. Maybe transpose it back, so just a lambda prime. Usually, you would transpose it, right?

FRANK SCHAFER: OK. Then it's f.

STEVEN JOHNSON: Just transposing both sides of that equation for lambda.

FRANK SCHAFER: gu, right?

STEVEN JOHNSON: gu transpose. And there's an initial condition, right, so for--

FRANK SCHAFER: Yeah, so that has an initial condition. Right, the initial condition is kind of important because we see, the first ODE, we solve forwards in time, and then the adjoint ODE, as you are now used to probably from reverse-mode AD, we have to solve backwards in time. So we have also this forward-backwards workflow, and then once we know u and lambda, we can actually solve the integration for dG dp.

STEVEN JOHNSON: Can you write that out? Just because it simplifies now, so several of the terms have dropped out.

FRANK SCHAFER: Oh, just g--

STEVEN JOHNSON: And write big.

FRANK SCHAFER: OK, that's a--

STEVEN JOHNSON: No, no, wait, there's no-- I thought you wanted to write it in terms of lambda, right?

FRANK SCHAFER: Oh, yeah.

STEVEN JOHNSON: Yeah, there's no s anymore.

FRANK SCHAFER: Wait. Yeah, exactly. And g-- yeah, that should be right.

STEVEN JOHNSON: And then plus a boundary-- oh, no, the boundary term is 0 in your case.

FRANK SCHAFER: Yeah, I set it to 0.

STEVEN JOHNSON: So before you continue, let me just compare this. We talked about adjoint methods in class before, but for solving, like, linear systems or nonlinear equations where the dependence on parameter, and it was of the same flavor.

So first, you solve the forward equations. You solve ax equals b for x, right, and then we solved an adjoint equation that involved a transpose or a transpose of a Jacobian, right? But whatever the adjoint equation was of the same kind of-- same flavor and same size as the original problem, but it was linear, right?

And then once you have the forward solution, the adjoint solution, you just take some dot products. So here, you solve an ODE, right, and you're using whatever your favorite ODE solver is. Then you solve an adjoint ODE, which is sort of transposed, and it's linear now, even if the original one was linear, and it's backwards in time if the original one was forwards in time.

And then once you have the forward solution u of t and the adjoint solution lambda of T, you take some dot products, which for functions, the analog of that is integrals, right? And so everything in that sense becomes easy now. Independent-- even if you have 10,000 parameters, you solve one forward ODE, one adjoint ODE of the same size, and then some dot products. So this is why you can compute derivatives with a huge number of parameters efficiently, and they're kind of always of the same flavor, these adjoint, reverse-mode things.

FRANK SCHAFER: Yes, and now there are a few tricks, basically, to evaluate that efficiently, or different styles to evaluate that equation. The first thing to notice is basically that we don't necessarily, for this continuous adjoint method, have to store and compute the full continuous solution of u of t because an ODE is basically reversible. And so we can solve the ODE forwards, just save the final value of that ODE, and then solve it basically in lockstep backwards together with lambda to have-- well, to avoid this kind of storing the continuous solution of u of t at all times. And everybody happy with that kind of trick, right? So--

STEVEN JOHNSON: I'm not happy with that kind of trick, like, if it's chaotic or something like that.

FRANK SCHAFER: But ODEs are reversible--

STEVEN JOHNSON: Solving forwards might be exponentially decaying, and solving it backwards might be exponentially growing. So that trick doesn't always work, right?

FRANK SCHAFER: Yes, exactly. That trick has to be done with care, and there are some-- right, so that trick, just to visualize it again, it basically says, if we have some ODE at u of t0 and we solve it to some final time, u of capital T, I was saying, you can just store this value and start solving it backwards. But depending on your physical system, well, your trajectory might not-- because your numerical solver might not be reversible, you might deviate from your true solution.

And then if you solve it for long times, again, backwards, that might be numerically very bad, and so the first thing people are usually doing is introducing so-called checkpoints, which only means, once you solve it back and you reach your checkpoint, you basically reset your solution to the state which you had forwards. So that reduces a bit the memory consumption. And the second trick people are doing, if it's really unstable and your ODE solver is not working, then what you would do is maybe also have these checkpoints but always just store a continuous solution within the checkpoints that you solve forwards again in time.

STEVEN JOHNSON: So just another comment on that, so one thing we have to be always careful about in presenting things is explaining the solution before we explain the problem, right? So the issue here that you're trying to solve with going backwards and reversing and so forth is-- so in principle, you solve it forward, right, to find u of t, and then you solve it backwards to find lambda T, and then you take some dot products, right? The problem you can run into is that then you need to store u of t not just at the endpoint but at all the intermediate times, which is fine if u of t is just a small number of parameters.

And you just store it at a bunch of points, and then you basically can interpolate with polynomials in between. Right, this is how ODE solvers work. But this is a problem if you're solving a huge system of ODEs, where, say, it's really a discretized PDE, for example. So you have u is, say, the electric field at every point in space as a function of time in some big computational cell, then solving-- then usually when you solve it, you march it forward in time, and you kind of just only store the solution at the current time.

But here, you need to store all the solutions at all the intermediate times all the way until the end and then go backwards, and you can run out of memory. And so this is the basic problem with adjoint methods and when you have, like, evolving in time or going through big recurrences, is you need to store all the intermediate ones. And so there's all these tricks to try and only store some of the intermediate values at some checkpoints and kind of reconstruct things on the fly as you integrate lambda T backwards so that you don't need to store everything, but that's sort of the issue that you're struggling with here. But for small ODEs, to only have a few things at any time, you just store the whole solution. It's not that hard.

FRANK SCHAFER: Right. True. Yeah, well, for small ODEs that contain some parametric function, that's very expensive to evaluate, right? Like, what Chris is basically doing is putting a neural network inside the drift function, so even if the ODE itself looks very simple, if you have a neural network in each timestep, you will have to store all intermediate computations also, and outputs of that. So, yeah, that makes the continuous adjoint very meaningful.

Yeah, right, so there's a third kind of trick which can be used to simplify that workflow even a bit more. Basically, that's the first trick, and as we now discussed, it's not always applicable. But if it's applicable, it can be nice and very efficient.

The third trick is basically to look at this integration term and basically realize that an integration can also be phrased as a solution to an ordinary differential equation. So I'm not sure if that was covered or-- but, yeah, let's say I have a function F. I want to integrate it as [INAUDIBLE] prime to delta t.

But everybody agrees, that's kind of the same form? So we have some integration, and now the claim is that we can write that as an ordinary differential equation small f of t. And, yeah, now we want to basically apply Euler's method to that, and so if we apply Euler's method--

STEVEN JOHNSON: So, wait, stop here for a second. So before you again give the solution, so you're going to write a differential equation w prime equals F of T, right? So wt is going to be a new variable. What's the initial condition of that, that you're going to want to use?

Again, at the end, you're going to want to write down a differential equation whose solution gives that integral, so what-- you're going to write a differential equation dw dt equals little f of t. What initial condition should you use? Any guesses? To get-- to turn it back-- so its solution will give you the same thing as that definite integral from t0 to t.

AUDIENCE: [INAUDIBLE].

STEVEN JOHNSON: 0, OK, so but at what time?

AUDIENCE: t0.

STEVEN JOHNSON: At t0, so w of t0 equals 0, and then what's-- capital F of T equals what, then? So now the integral you want, the first line, capital-- is that an I or an F? I can't--

FRANK SCHAFER: That's an F. Sorry.

STEVEN JOHNSON: That's a capital F, OK.

FRANK SCHAFER: Sorry.

STEVEN JOHNSON: It has the second line going kind of-- looks a little bit-- OK, so anyway, so your capital F of T, your integral that you want is what in terms of w? So if you have-- the differential equation, its initial condition t equals 0 is 0-- t0 is 0. Its derivative is little f of t, so then the integral is what now?

AUDIENCE: [INAUDIBLE].

FRANK SCHAFER: Yeah, right, so I can write that integral. Let me write one step in between, basically, and kind of slide it.

STEVEN JOHNSON: You're kind of discretizing it first, so before you discretize it, I want to make sure they understand the connection between the ODE and the integral. So don't discretize it yet, all right? Just, this is an exact relationship, right?

FRANK SCHAFER: Oh, so you just want to have plugged in the w, so--

STEVEN JOHNSON: Yeah, yeah, it's w of what? No, there's no integral. It's equal to w of-- it's w of capital T, right? That's--

FRANK SCHAFER: Oh, right.

STEVEN JOHNSON: Right.

FRANK SCHAFER: OK, now I see what you want you want me to write.

STEVEN JOHNSON: Right. So the point is, if you have a differential equation w prime equals f of t, the solution is just the integral. w is just the integral of f, which is what you want, and then you put the initial condition at t0 equals 0. That's the left edge of the integral, and then the final integral you want is just w of t. So that's-- he's turning-- the integral you want is just a differential equation, is this differential equation, and now you're going to discretize it, right?

FRANK SCHAFER: Right, now if I'm discretizing it, I will see the same, basically, as for-- yeah, for an Euler step. I should write it down, or--

STEVEN JOHNSON: Sure, if you want to discretize it. Yeah, I just wanted to-- because not everyone here has done a lot of differential equations, right, or maybe hasn't done them recently, so I want to make sure they understand, understood that trick. It's good. Take it all the way up now.

FRANK SCHAFER: Oh, OK.

STEVEN JOHNSON: [INAUDIBLE].

FRANK SCHAFER: Right, so I'm basically writing down what the Euler steps would be in the discretization for that integral. That's the first, and then I just keep iterating. Let's think a moment for--

So these are n Euler steps iterated on the function. There's an i missing, probably, somewhere. Right, because once I have my first step, the time changes, and then that's flagged back also into the function.

STEVEN JOHNSON: OK, so this is really an approximation for the original thing, right? So you're writing "equals" here, but just to be clear, right, the original equation is-- you have an integral and the differential equation. That's an exact relationship, and now you're going to write the discrete analog of that, or the discrete approximation. So w of t plus delta t is really approximately equal to w of t plus t delta t, and this is approximately.

FRANK SCHAFER: Yeah, if I do it again-- something like an Euler step.

STEVEN JOHNSON: So you can approximate the derivative by a difference, and then the integral of your-- is essentially approximated by a sum.

FRANK SCHAFER: Right, and so this now has, then, also, I mean, the typical form of what you would do numerically to solve an integral, right? So it's-- yeah, you have the function value at different points, and then with the discretization that you use and summed it all up.

STEVEN JOHNSON: This is a very simple way to do an approximate integral numerically, just like-- Euler integration for ODEs and this approximate sum for an integral, you should never, ever do this in reality, almost, because there's exponentially more accurate ways to integrate approximate integrals. But it's a nice way to really simply think of-- if you're confused by integrals and derivatives, you can think of them as differences and sums or sums and differences, right?

FRANK SCHAFER: Yeah, right, that statement is true for both, right, the Euler solver for ODEs and the kind of derivation for the integral. Yeah, that's actually the full derivation, I'm sure.

STEVEN JOHNSON: Yeah, so does everyone see that basically, then, if you plug in, the initial t is 0. w of t0 is 0. Right, so and then why don't you take it one step further?

So then w of t0, if w of t0 is 0, the first term is 0. The second term is approximately the integral, right, so it's just a restating the same thing, just with sums and differences. So this is-- so the solution of the ODE is really the integral.

FRANK SCHAFER: Yes. Yeah, to summarize that, so we can have, then, basically all the three methods above.

STEVEN JOHNSON: So can you finish and write-- so the point at the-- what you're going to do then is just merge the solution of the adjoint ODE with that integral in the third step, so instead of doing them in separate steps, so you can maybe just write down the derivative. It has a--

FRANK SCHAFER: That one, right?

STEVEN JOHNSON: Yeah, so what's w? w is now going to be-- no, no, w is going to be lambda and-- OK, it doesn't matter.

FRANK SCHAFER: Yeah, depending on the other side of--

STEVEN JOHNSON: You want to go in the other direction, right? Yeah.

FRANK SCHAFER: Probably want to solve it like that.

STEVEN JOHNSON: But the point is, you think of this as a coupled ODE, for w along with the adjoint ODE for lambda prime. So if you write down the lambda prime underneath, right--

FRANK SCHAFER: gu.

STEVEN JOHNSON: So you can think of this as one system of ODEs for a vector that contains w and lambda, right, so you just do the two things together instead of the separate steps. Oh, and then I guess, yeah, you could also integrate the original equation backwards as well. All right, so now your unknowns are w, lambda, and u, so, like, one big vector of those three things stacked together.

And the advantage of this is that you don't need to store the lambdas, I guess, and maybe the u's. If you were able to integrate the whole thing backwards, you don't need to store anything. You just pass this to your ODE integrator, and it only keeps track of your current step. And it just walks from beginning to end, and at the end of the day, your w is the integral that you want-- is the derivative.

FRANK SCHAFER: Yeah, so that's the most memory-efficient one to basically solve for the sensitivity dG dp in the end. Right. That's all for--

STEVEN JOHNSON: So I think we should maybe take a break for the-- or in a minute, but are you basically-- is this basically it? Or are you going to show some examples?

FRANK SCHAFER: I would have the integration of that numerically, but it takes another 10, 15 minutes at least.

STEVEN JOHNSON: Yeah, I see you want to show an example of-- yeah, I mean, we can do that after the break, but maybe we can just take a break for a couple minutes now. But do people have questions before we-- about this step? You have forward ODE, adjoint ODE, and then a dot product, which is basically the integral. And then there's these tricks to try and basically combine them into one ODE that you solve instead of having to store u of T, store lambda of T, and then do the integral.

AUDIENCE: So say you have some data point that you're comparing to something you're looking for, would you-- so if the system's not chaotic, you'd probably want to start it at the final data point and integrate backwards with the lambda equation, but if it is, then would you just want to integrate forward to get the full solution and then just solve that?

STEVEN JOHNSON: Yeah, so suppose capital G-- wait, so suppose capital G is a sum and not an integral, right? So you're-- or you have a bunch of delta functions, is another way of thinking about it, right? So basically, you're comparing the solution u of T to, say, some known solution, some measured data, at only a discrete set of times. So now G is a sum, right, so one way to do it would be to compute u of T, lambda of T, and then just do the sum. But is there a way to combine that into a single ODE if now capital G is not an integral, it's a discrete summation over a set of times?

FRANK SCHAFER: Yeah, basically, you can write it-- go to this kind of discrete formulation. Probably the easiest way, I think, to think about it is that these describe the kind of continuous changes that you want to have in the cost function, and so what you will have if you just have your cost function compared at discrete time points you-- right, you basically have your solution.

And then at some point, you compare your cost, and so what will happen is that you just solve-- you will have a backward solution starting from here for the adjoint and for u. And then at these time points where you have your comparison, you basically have to-- you can update your state by the change that comes in just from those nodes, basically. So we'll just have, like, partial derivatives added at those time points to your adjoint state.

STEVEN JOHNSON: Yeah, so, I mean, ODE solvers, you can tell them to stop at certain times, so as you go backwards in time, you can just have it stop at that time and then update something and then stop at that time and add-- at the next time and add something to accumulate your sum as you go along. But mathematically, you could basically think of it as jump conditions that occur at those times, and so there's also ways to include those in differential equation solvers, where, oh, every once in a while, these times, something jumps. There's a discontinuity in the solution, and then in between, it's not changing at all, in this case.

FRANK SCHAFER: Yup.

STEVEN JOHNSON: Probably, it's easier just to say let me integrate it up to this time, get that solution up to the next time, and just have a callback at those discrete times and just accumulate the sum.

AUDIENCE: One more question about that, so maybe cycling back to the idea of matrix calculus, so here, everything is still-- basically, if you want to generalize it, say you have a matrix-valued ODE, then your lambdas become some matrix-valued thing. So they live in the dual state to whatever you want to take the derivative of, and as long as it sort of makes sense, the ODEs that come out at the other end, that will give you the right sensitivity? Is that--

STEVEN JOHNSON: Yeah. Yeah, so if you have a more general space-- so u, u lives in some arbitrary vector space, right, then you have to replace the lambda transpose there with an inner product, right? And then this fu transpose is some more general adjoint operator in that Hilbert space, right, so all of this still works. So you can think of this just-- it's literally the same derivation. You can think of it as just a change.

It's just, lambda transpose is now notation for that inner product. It's just-- you have to understand what the notation means in a more general sense, but it's literally the same, same thing. I don't know, have you done anything where-- I guess you're doing quantum systems, so probably, your states live in some more--

FRANK SCHAFER: Density matrices, yeah, sure.

STEVEN JOHNSON: --live in more general Hilbert spaces all the time.

FRANK SCHAFER: Yeah. Yeah, and numerically, you can also then vectorize the density matrix, right? And they have-- yeah. Yeah, rewrite, basically, also the definitions of f to deal with vectors depending on your implementation, but, yeah, right.

STEVEN JOHNSON: Yeah, so we talked about that as well. So one way of dealing with matrices and other things is just to turn them into column vectors, right, but if it's more natural in your system space to talk about matrices, then you'd really rather deal with that Hilbert space directly.

FRANK SCHAFER: Right.

STEVEN JOHNSON: And if these are PDEs, so u is now not-- is like a function of space, right? Of course, on a computer, you probably discretized it, but it's nice conceptually to think about this as a function of space and this inner-- lambda is also a function of space, for example, and then this inner product is some kind of integral over a space. So it is really powerful to be able to not have to turn everything into a column vector when you want to do linear algebra.

Got a five-minute break, so we'll come back at 12:10. And I could go then, or if you want to show some more examples, you can do that. But--