[SQUEAKING]

[RUSTLING]

[CLICKING]

**STEVEN JOHNSON:** So I wanted to start the day just by finishing up a couple-- comment on a couple of additional things from the last lecture. So last lecture, we talked about finite difference approximations. In particular, I talked about the most obvious approximation, which is just from the definition of the derivative. We approximate df by f prime of xdx, our linear thing, the exact derivative for a infinitesimal, dx, by just f of x plus dx minus f of x for some finite delta x, noninfinitesimal.

And if you divide both sides by delta x, the more familiar form is was this one. This is called the forward difference approximation . And we showed that this has an error in it that decreases linearly with dx until dx becomes really small and then becomes limited by roundoff errors.

But I wanted to mention that, of course, there are other ways of approximating derivatives by differences, some of which are even more accurate. They're called higher order finite difference. So the forward differences are first order because the error goes linearly with dx. The most famous next improvement would be approximate f prime dx by instead of f of x plus dx minus f of x, we take f of x plus dx minus f of x minus dx.

And since now you've basically doubled the interval, because you're going from minus dx to plus dx, you have to divide by 2. Or if x is a scalar, so then you can just divide by delta x. The second form is the more familiar form of these center difference approximations, which is you divide both sides by delta x, which only works if delta x is a scalar. And you get f prime is approximately f of x plus dx minus f of x minus dx divided by 2 dx, so again, 2 because you're going from minus dx to plus dx. So the overall interval is 2dx.

So it turns out that the error in this case, instead of going linearly with delta x, is quadratic. I guess this should be order delta x squared. I left out the x. I'll fix that later. And the reason it goes quadratically-- so this is called second order accurate-- you can derive it in a variety of ways. But one way to derive it is to just plug in the Taylor expansion for f of x plus dx and f of x minus dx.

And what happens is because this is symmetric-- it's f of x plus dx minus f of x minus dx, all the terms with even powers of delta x in the Taylor series all cancel because if it's an even power, like remember, the there's f plus f prime dx plus f double prime dx over 2, and so forth-- that's your Taylor series-- f double prime dx squared over 2. Sorry.

Anything with an even power of delta x, It doesn't matter if it's plus dx or minus dx, it's the same thing when you square it or take the fourth power and so forth. So when you subtract these two terms, they cancel. And so the error becomes the next order term, assuming it's at least three times differentiable.

So it turns out this center difference formula, the error and the derivative goes like delta x squared, or if I multiply both sides by delta x, the error in the difference goes like delta x cubed. And we can check this. We can plot it for sine x. So I plot, for sine x, we're going to approximate it at x equals 1 by either the first order difference, sine of 1 plus something minus sine of 1 over s, or the center difference, sine of 1 plus s minus sine of 1 minus s over 2s. I don't know why I called it s there.

And I'll plot it as a function of delta x. And the forward difference is the blue. That goes linearly with delta x. The blue line is just a plot of delta x for reference, linear. And the central difference is the red dots. They go quadratically with delta x. And the straight red line is just a power law quadratic with delta x.

Of course, again, this is a log log scale. So power laws turn into straight lines until in both cases, once they hit the machine precision, the error starts getting worse, if you make delta x too small. And in both cases, you want to have a delta x that's small, but not too small, like 10 to the minus 5, in this case, for the quadratic case, or 10 to the minus-- yeah, 5 was about where this-- the error is minimum in this case.

And in fact, there are even higher order differences. If you take even more points-- in some sense, what you're doing is basically taking multiple points and fitting it to a polynomial and taking the slope from that. So as you get to add even more and more points, you can fit to higher and higher order degree polynomials and get higher order difference formulas that give even smaller error for the same size of delta x.

Another even fancier thing to do is you can start with just some delta x and then compute a sequence of delta x's that you keep dividing it by 2 or something. And each time you divide it by 2, you fit it to a higher degree polynomial. And you're extrapolating the slope to delta x equals 0.

And it turns out there's a way to do this adaptively. So you keep decreasing delta x by, say, a factor of 2, and you go to higher and higher order polynomials. But there's a way to watch this and figure out that at some point when delta x goes too small, you can notice that the error is getting worse. And you can stop. So you can adaptively find the optimal delta x and the optimal order to extrapolate as accurately as possible.

And the fancy version of this is called Richardson extrapolation. So it's a really nice technique for extrapolating functions. I'm not going to go into it in more detail. But it's a fun thing to look up. I have a package for it in Julia. And there's a Julia package called FiniteDifferences.jl that has zillions of different finite difference formulas, including Richardson extrapolation. They can get pretty high order accuracy.

But anyway, so I wanted to mention those. And the other thing I wanted to mention was what happens when you go to higher dimensions, so higher dimensional inputs. So suppose instead of just x is a number, you have x is a vector or something like that. So of course, this formula still works. We can still approximate f prime dx by f of x plus dx minus f of x. And there are still higher order versions as well. We can no longer divide by dx. But that's not a big deal.

But each finite difference, if x is a vector, each finite difference basically only gives you the derivative in one direction in that space. So think of, for example, if f is a scalar, a scalar valued function, and x is a vector, then what you really want is the gradient. And every component of the gradient is the derivative in one direction in that space.

But every finite difference only effectively gives you one component of the-- can give you one component of the gradient. It can only give you the derivative in that direction, dx. So if you want the derivative in all possible directions, and x is n dimensional, you need n finite differences. So for example, if x is an RN and f is a scalar, and you want every component of the gradient, and it has n components, you want each one of those partial derivatives, each one of those is a separate finite difference. So you need n finite differences.

And so what happens is this gets this gets very expensive. So if you have three parameters, fine. You need three finite differences. But if you have a million parameters, then you need to take a finite differences a million times. You need a million different delta x's. And if f is a very expensive function, like evaluating a neural network, this is horrible. Evaluating a neural network a million times is very expensive.

So the net result is that in high dimensions, this becomes incredibly impractical to compute finite differences if you really want all the derivatives, the derivatives in all directions, which you want for things like optimization. So you really, really need to do derivatives analytically in high dimensions in order for this to be practical for things like machine learning, optimizing neural networks, to be practical. That's the only way to get the a million derivatives efficiently.

On the other hand, they're still very useful as a check. So as a check, if you're developing your own derivative, for example, if you're doing your homework problems, and some of those functions, it's not so easy. Derivatives-- you might think in 1801, derivatives are easy. But when the functions get more complicated, you have vectors in, vectors out, or vectors in, matrices out, probably you're seeing it's not always so easy to get the derivative and be confident you did all the chain rules and product rules correctly.

So it's really, really useful to check it. And I definitely encourage you on homework to check it. And you can very easily check it just by, basically, just computing a few random directions, so to choose delta x as a random vector that's small, do it a few times. If that matches to several decimal places your f prime dx, that's a pretty good sign that you didn't make a mistake. It's very unlikely that if you have a bug, that it will just happen to be correct to six decimal places in some random direction in space.

So I just wanted to stop with that. So any questions about finite differences before I'm going to move on to a completely different topic?

So what I want to do now is talk more about gradients. And I want to generalize gradients to higher dimensional problems. So the familiar gradient from 18.02 is when you have a function that takes a vector in and gives you a scalar out, just where the vector is just like an n component column vector. And then the gradient is an n component column vector. It's a component vector of partial f, partial x1, partial f, partial x2, and so forth.

And in this class, we try and think of it more holistically that we have the differential, df, is f of x plus dx minus f of x. So this is just review. Our derivative is the linear operator that takes in the dx and gives you df to first order. So this linear operator takes in a vector dx and gives you out a scalar. So we sometimes call that a linear form, something that takes a vector in and a scalar out.

And if this is a column vector, an n component column vector, it's pretty easy to see that the only linear operator that takes a column vector in and gives a scalar out is multiplying by a row vector, or equivalently, is taking a dot product of dx with some vector. And that thing we call the gradient. So this f prime of x is then a row vector. It's the transpose of the gradient and the gradient, or equivalently, the gradient is the thing you take the dot product of dx with to get df. And this is equivalent to the 18.02 definition of gradient.

But what I want to do today is talk about generalizing this to other kinds of vector spaces, to where x is not necessarily a column vector, but some other kind of vector. But we still have a scalar valued function. So we're still going to generalize to-- we're still talking about scalar value functions, for example, a function that takes a matrix in and gives you a number out. So for example, a determinant is a function that takes a matrix in, but gives you a scalar out. And can we talk about the gradient of such a function? What is the gradient of a determinant?

And so we'd like to have it this same kind of thing, where it's some kind of dot product, and where the gradient is of the same shape as x. So if x is a column vector here of n components, the gradient is a column vector. And this is going to be really important soon, when you start talking about optimization because then you can think of the gradient as the uphill direction for the function. So if you want to make the function bigger, you go in the direction of the gradient. That's uphill. If you want to make a function smaller, you go in the opposite direction of the gradient. That's downhill.

So how do we generalize this to other kinds of things that are not just n component column vectors and not the familiar 18.02 gradient? So I think it's pretty easy to-- what we basically want is the same kind of thing. If x lives in some arbitrary vector space, and we have a scalar function that takes x in that vector space and gives you a number out, a real number-- everything's real in our class right now. So all our scalars are real numbers.

So it's a linear operator that takes a dx, a vector, in and gives you a scalar out. And we want to define a gradient. What we want is we want to define this as some kind of dot product. And so to do that, we need to define a dot product for our vector space. So in general, gradients, we're going to be able to define gradients any time we have a vector space, we have a scalar value function, and we have a dot product on the vector space. So we need to generalize our notion of a dot product to other kinds of vector spaces.

So first of all, let me just give you the general definition of a dot product. So usually, once you generalize it to other kind of vector spaces, you don't call it a dot product anymore. People call it an inner product. But dot product is a perfectly good colloquial name for that. But the pure mathematicians will look at you sideways a little bit if you call it a dot product. You call it an inner product.

So an inner product, or a dot product, is some rule that takes two vectors in our space. Call it x and y. And again, this is some general vector space. So it could be column vectors. But it could also be matrices or something even more complicated. And it's a rule that gives you a scalar out of them. And we can denote that in 18.02 style, or freshman physics style, as x dot y. That's there.

More commonly, in pure math, they often use angle brackets. So they would denote this thing by angle x, comma, y. I just wrote it here. In quantum mechanics and physics, they like this bracket notation, where they use angle brackets, but they put a vertical bar in between. And there's some complications there. They really call the vectors, they call them x, with brackets around them. But I'm not going to worry about that too much. But you may have seen, this is an inner product. This is an inner product. This is an inner product. I'm just going to use dots in this class, just to keep it kind of elementary looking.

So it's a rule that takes two vectors and gives you a scalar out. And right now, we're dealing with just real vector spaces. So our scalar is going to be real numbers. And it has to have certain properties to be a dot product. Basically, you need to obey certain rules, so that you can do the familiar rules of algebra with it that you do with your familiar dot products. You want it to work, those algebraic rules to work, with your dot products on whatever crazy vector space you have.

And it turns out, you really just need three rules. So first of all, we want it to be symmetric. So the rule should be that it should satisfy that x dot y equals y dot x, for any x and y. If these are complex numbers, then you need a complex conjugate over this. But since this is real numbers, I'm only dealing with real numbers, I can just say it's symmetric, but just keep in mind, if you ever work with complex numbers, there is a conjugation here.

Second, and of course, our familiar dot product certainly is symmetric, where you multiply the components and add them up. That's symmetric. And then it has to be linear. And so that means if, for example, if you take the dot product of x with alpha y plus beta z-- and as usual, Greek letters mean scalars here-- then that has to be the same thing as alpha times x dot y plus beta times x dot z. And again, the familiar dot product is certainly linear.

And since it's symmetric, this also means it's linear. I wrote it down here, it is linear in the second argument. But from rule number one, I would swap the second argument for the first argument. It means it's also linear in the first argument.

And the third rule that it has to satisfy if we want to call it an inner product is that it has to be non-negative. So if you take the dot product of a vector with itself, we're going to call that the norm of the vector squared. So the norm of the vector is the square root of this. So x dot itself is norm squared. Just like the familiar dot product, the dot product with itself is the length of the vector squared. That length had better not be a negative number.

We're going to require that anything, if it wants to call itself an inner product, x dot itself has to be non-negative. And furthermore, it can only equal if and only if x equals 0. So the 0 vector-- every vector space must have a 0 vector. And the 0 vector, its length had better be 0. And it better be the only vector that has a length of 0. And if you have these three rules, then you can call it an inner product. And any questions on these rules yet? And I'm going to give some examples in a second.

So this is an inner product. And so now, just for terminology, I'm not going to use this terminology much, but if you have a continuous vector space-- you know what a vector space is. So if you add an inner product and it's a continuous vector space, so it's the real numbers and you can vary the vector smoothly, people call this a Hilbert space. So if you ever hear that term, don't get scared. It just means you have a vector space and you have a dot product.

And so now we can define a gradient. So if you have a scalar function that takes a vector in and gives you a scalar out, but vector space has a dot product, so it's a Hilbert space, then the derivative must be a linear function that takes a vector in and gives you a scalar out. And it turns out that if you're in a vector space and you have a dot product, and you have a linear function that takes-- it's called a linear form that takes a vector in and a vector out, every linear form must be of the form the dot product of some vector with the x. This is always true.

So this is certainly true for column vectors. For a column vector, if it takes a column vector and gives you a number out, it must be a dot product with something. It turns out this is true in general for any Hilbert space, any vector space with an inner product. If you have a linear function, vector in, number out, it must be a dot product with something.

This is called the Riesz representation theorem, by the way. I feel like I spelled Riesz wrong. I think it's C-Z, like this. Yes, I think it's C-Z. It's a Riesz representation-- I'll double check. Or is it Z-C? No. I think it's C-Z, one of those names. I can never remember it. Yeah, so this is the theroem called the Riesz representation theorem. But it's kind of an intuitive thing.

If you think of column vectors, it's very intuitive. If you have a linear rule, a column vector to a number, it must be a dot product. It turns out this is always true. This must be a dot product with something. And we call that thing the gradient. So the gradient for any vector space, any scalar function of any vector space with an inner product, is the thing you take the dot product with to get your df. So this is our df.

And so it's always something that has kind of the same shape as x. So let me just do some examples. So of course, the example, the familiar example, is just our vector space is just our n. So it's n component column vectors. And then our dot product is just x dot y is x transpose y. That's our familiar dot product.

Well, I shouldn't say therefore. this is the familiar Euclidean dot product. It's not the only possible dot product-- Euclidean. There are actually other dot products. You could also have this. So this is-- what is this? This is basically x1y1 plus x2y2 plus dot, dot, dot. You just multiply the components and add them up.

But you could also have, for example, a weighted dot product. So we have-- let's call it x dot-- I need to use a different symbol. Let's call it x dot sub w y is-- I could do x1y1-- sorry-- w1x1y1 plus w2x2y2 plus dot, dot, dot for weights w1 to wn that all have to be positive.

Why do they have to be positive? Because of the non-negativity property, for us to call this an inner product. Let me put those in black. If it had property property 3, you have to have dot products be non-negative. It means the weights will have to be non-negative.

This is a perfectly good inner product. It's not the Euclidean one. It weights each dimension differently. But this might be useful in statistics or something like that. For example, if you want to have a bunch of measurements, but there's a different uncertainty in each measurement, you might want to weight the numbers-- the components that have more uncertainty, you might want to weight them less. You give them less weight. And the ones that are more certain, give them more weight.

Or another example, if you had a vector where the different components have different units, for example, a vector where the first component is in meters, and the second component is in seconds, and the third component is in kilograms, then the ordinary dot product formula doesn't even make sense. It'll be adding up things with different units.

So then you have to weight each one. You divide-- the first weight would be 1 over some characteristic length scale in meters. The second one would be some 1 over some characteristic length scale in time. The third one would be 1 over some characteristic length scale in kilograms. And that way, each term is dimensionless. You can add them up or something like that.

Another way it's also useful to write this down is linear algebra. So we could also write this down as x transpose y. And what do we put in the middle to get this? Any ideas? What should it look like in the middle? What can I put here? Let me give you a hint. It's a matrix. What's the entries of that matrix? Alex, if someone says an answer, I can't hear anyone in the room. Alex, is anyone raising their hands?

**AUDIENCE:** Yeah. W on the diagonal. Can you hear that?

**STEVEN JOHNSON:** Exactly. I cannot hear anything in the room. So you have to speak into the microphone. So that's your job, is to repeat any comments or questions from the microphone. Yes. So it's just a diagonal matrix. And an another example is another weighted dot product.

Let's call this x-- so this was with-- let's call this x dot y sub capital W, for column vectors, would be x transpose y. And you put any matrix in here. But you need it to be symmetric for property one. So we need W to be symmetric for property one. And we need for property three, for positivity, we need W to be positive definite.

And so the one we had before is just a special case of that. A diagonal matrix is certainly symmetric. If it has positive diagonals, it's certainly positive definite. But you can have some arbitrary thing that mixes them up. And these all are useful in surprisingly-- so there's a surprisingly large number of cases in which you don't want to have the traditional Euclidean dot product. You want to have something more complicated.

And with all of these things, you could then define gradients differently. So if you have a weighted dot product, it's going to change your definition of the gradient because it's going to be the thing with that W in it. But most of the time when we're talking about gradients, we're talking about this is the-- you're talking about the familiar one. The Euclidean product gives you the usual gradient. But you might need a weighted dot product, for example, for grading to make sense if you have a derivative with respect to a bunch of different components that have different units.

So let's do another example that's even less familiar, though. Let's say that v is R m by n. let's do m by n. So this is going to be m by n matrices. Those form a perfectly good vector space. I can take two matrices, and I can add them. They have the same shape. I can multiply them by 2 or some other scalar. I can subtract. So this is fine as a vector space.

And in fact, using that vec operator that Alan talked about in the last lecture, you could convert it into column vectors-- you can stack them up into column vectors with m times n components. But I want to leave them as matrices. But yeah, let me mention that because this has come up in a second.

So there's an isomorphism with this vec function to column vectors. Vec, vecA that live in R m times n because this is really an mn dimensional space. You could just take all the entries of the matrix and stack them up. That's what vec does, if you remember from last lecture. This just stacks the columns.

And I don't know if people know the term isomorphism. How many people know what this means? I don't know if people are raising their hands or not. Alex, are people raising their hands.

**AUDIENCE:** Five people.

**STEVEN JOHNSON:** Yeah. So it basically means that if you add two matrices, it's equivalent to adding these two vectors because-- or multiply by 2. It's multiplying these vectors by 2 because these vectors are just the same component entries of A rearranged. So adding matrices is adding the entries. Adding these vectors is adding the entries. Multiplying by 2 is multiplying the entries by 2. Multiplying this vector by 2 is multiplying the entries by 2. So it's the same thing.

But anyways, and sometimes it's useful. It's nice to stick with-- if your problem is expressed in terms of matrices, usually you want to keep in terms of matrices if possible. That's the natural way of looking at your degree of freedom is an n by n matrix, whereas stacking it up into a column vector of mn components is kind of an unnatural way. Internally, the computer does that. But sometimes it helps you disentangle what's going on if you stack things up into a more familiar vector.

What we'd like to do is define a dot product of matrices. And again, just like for vectors, there'll be multiple choices. There's never one dot product for a vector space. There's multiple possible choices. And you choose one. And then everything else follows from that. The geometry follows. So I just want to show you the most obvious one, the analog of the familiar Euclidean inner product.

So what would that be? So if I have two matrices, A and B, where this is m by n, and this is m by n, And this is our dot product. This is not this is not a matrix product. This is our inner product. So we want to give this to be a scalar. what should I do? Any suggestions? Alex, you'll need to repeat. I can't--

**AUDIENCE:** Yeah. This one, multiply by all the components, like elementwise.

**STEVEN JOHNSON:** Exactly, just multiply the elements, the entries of these matrices elementwise, and add them up. So this is the sum of the elementwise products. So that's it's a little bit annoying to write that out in words. So let's see if we can figure out a better way to algebraically to write that down.

So in Julia, you could write this down. The one way to do this is to say sum of A dot star B. Dot star is an elementwise product of two vectors and matrices. And sum just sums then sums all the entries. That's what this is. I could write it as sum over i and j of Aij Bij.

I could write it with vex. It's the same thing as if I take the elements of A and I stack them up into a vector, and I take the elements of B and I stack them up into a vector. This is just the ordinary dot product of those. VecA is just the same numbers. It's just stacked up. So if I take the ordinary dot product of vecA and vecB, that is this elementwise product.

But again, that that's a little bit frustrating because if someone hands me this vector space, it's probably because they really want to leave things as matrices. That's the natural language of the problem. So I'd rather use this-- this vec notation is kind of a last resort. If you can't figure out how to deal with it as a matrix, you cram it into a vector and then go back to what you know.

So it turns out that there's a really nice way to express this in linear algebra, using linear algebra operations. And that turns out to be the trace of A transpose B. I don't know if you want to if you want to see that derived. It's actually pretty straightforward to derive this. So this is also this is this is also called the Frobenius inner product of two matrices.

So if you just write out the formula for trace and matrix multiplication, you can actually see that. Another way of seeing that is basically, what is A transpose B? So A times B is take rows of a times dot product with columns of B. But if you do A transpose of B, A transpose, its rows are the columns of A. So what this really is, rows times columns here, is you're taking the dot product of each column of A with each column of B.

And the trace is the sum of the diagonals. So what's the first diagonal entry of this? The first diagonal entry is column 1 of A dot column 1 of B. The second diagonal entry is column 2 of A dot column 2 of B and so forth. And if you add those up, you're adding up all the dot product of each column of A with the corresponding column of B. That's exactly this. It's the sum of the product of all the entries.

So this is a really useful thing to know about, that you can take a dot product of two matrices. The obvious inner product is really the trace of A transpose B. And this then gives you the Frobenius norm, which I mentioned the other day, of matrices. So the norm of a matrix of anything, we want it to be the square root of the dot product with itself. And what this is is exactly the square root of the trace of A transpose A.

So it's really useful. It's kind of the obvious Euclidean norm of this. So this is exactly the same thing as the Euclidean norm of all the components, if we take and vecA and take it, to just stack up the components, it's that, or equivalently, it's the square root of the sum of-- if you just take all the entries of A and just take the square root of the sum of the squares. This is the most obvious way to measure the size of a matrix.

And so now, using this, we can now take the gradient of scalar functions of matrix inputs. So for example, let's start with the-- let's do some examples. Let's take some derivatives of matrix of scalar functions of matrices. So let's do f of A, where this is going to be an m by n matrix.

What's a scalar function? The simplest one is maybe the one we just learned a second ago. Let's do that. Let's take that Frobenius norm. This is the sometimes denoted-- sometimes people put an F here. I'm just going to leave out the F, but also-- let me just note that in because there are multiple ways of denoting the norm of matrix, just like there are multiple inner products.

So when people want to be specific, they usually put an F there. But I'm just going to-- well, I guess I can leave this here right, just so we know what it is. So this is the square root of the trace of A transpose A.

And now, I want to compute a derivative of this. So first, before I worry about the gradient, let's just make sure we can compute a derivative. We better be able to take what is df and use linear algebra, linear algebra. So how do we get df? So I need to use the chain rule. I want to use the chain rule, just to simplify life, so we don't have to rederive everything from scratch.

And so first of all, we're applying the chain rule to the square root function. But it's the square root of a number. Once you take the trace, this is just a number. And it's the square root of a number. So we can use the ordinary-- once you have scalar functions, you can use all your ordinary 1801 first year calculus rules. What we're doing is not different from that. It's just a generalization of that.

So you can really just use the 1801 chain rule to start with. So the derivative of a square root is 1 over 2, the square root of that thing, of trace of A transpose A times d of what you took the square root of, of trace of A transpose A. This is just the 1801 chain rule.

So we don't want to throw away everything we learned. So once we hit scalars, we can use our familiar rules. What we're doing is exactly equivalent to that right. So this is just the ordinary chain rule for square roots. It's the derivative, or the d in this-- the differential, in this case, of whatever is inside the square root times 1/2 that thing to the minus 1/2 power.

So let's see if we can go further. So now, so I want to do a-- so what's the d of the trace, of the trace of a function of, say, B. So this is trace of B plus dB minus trace of B. But the trace is linear. This is the same. The trace of B plus-- the trace of the sum of two matrices is the same thing as the sum of the traces because if you think about what a trace means, it's just the sum of the diagonal entries.

So if I add up two matrices, their diagonal entries just adds. That means their trace is just add. So this is just trace dB. So this is very closely related to problem 1 on the homework. So this thing is-- what do we have? We have 1 over 2. That square root is really just our norm of A in the front, so I'll just simplify that there. And then we have the trace. The d of the trace is the same thing as the trace of d of A transpose A.

And now, we can use the product rule. So this is 1 over 2 norm of A with a trace of-- and what do we have? We have dA-- that needs to be written-- dA transpose A plus A transpose dA. And now, I want to simplify this if I can, to try and move all the dA's to the right. So any questions so far? Again, Alex, if you have any questions, you have to repeat because I can't hear anyone in the room.

**AUDIENCE:** No questions.

**STEVEN JOHNSON:** Yeah?

**AUDIENCE:** No questions.

**STEVEN JOHNSON:** No questions, yes. So again, I want to move-- again, we're going to use linearity. So this whole thing is equal to the trace of dA transpose A plus the trace of A transpose dA. I'm going to try and make it look like a dot product with A, so I want to group my terms with dA so all the terms, the dA terms, are going to be on the right.

So one of the things I can use is-- so dA transpose A is not the same thing as A transpose dA. But its trace is. So note, over on the right here, I'm just keeping a running note of my properties. That was my note one. Note two, if I have the trace of B, that's the same thing as the trace of B transpose. If I transpose a matrix, it doesn't change the diagonal entries.

So this thing here is the same thing as the trace of A transpose dA. The matrix inside is different. But its trace is the same. B is not equal to B transpose, but trace B equals trace of B transpose. So that, of course, is the same as this. So I just have two of these. It cancels the two out there. And what you're left with is 1 over the norm of A times the trace of A transpose dA.

So this is quite a simple formula. The 2's cancel. And so what's that? That, I claim, is the dot product-- if we just go back to our formula, trace of A transpose B is a dot product of A and B. And so this trace here is exactly the dot product of A with dA. But then have a scale factor. But dot products are linear, so I can bring that inside. Let me move this up a little bit.

Does everyone see this? So this is, according to my definition of the dot product, this trace here is exactly the dot product of A over norm A, like the unit vector in the direction of, in some sense, with dA. And so this term here is exactly my gradient. This is my nabla gradient of norm of A is exactly A over norm of A.

So I've just taken the gradient of a normal matrix. And if you think of a vector-- actually, Professor Edelman derived this for if you took the gradient of norm of x, that was x over norm of x. You derived that last time. This is actually literally the same formula, which makes sense, because this Frobenius norm is really, in spirit, it's the same thing as your familiar norm of a vector, just the square root of the sum of the squared components.

So when you take the gradient, if you think of the gradient as the derivative of this vector, each component of A, you get exactly the same thing. And in fact, I should mention that. So in fact, in 18.02 terms, if you have a function, the gradient-- if you have a scalar function of a matrix for a scalar f of a matrix, this is exactly going to be the matrix of partial f partial A.

**GUEST SPEAKER:** Steven, just giving a quick hello to you and the class.

**STEVEN JOHNSON:** You're here. Great.

**GUEST SPEAKER:** Yeah, I just finished my talk. I was just giving a talk on Julia and high performance computing. It was a Dell talk. So anyway, yeah, I'm here. But let me not let me not interrupt you any more than I just have. Is the technology all working? Maybe I'll just ask that.

**STEVEN JOHNSON:** Yep.

**GUEST SPEAKER:** Good.

**STEVEN JOHNSON:** So it's exactly just the matrix of partial derivatives. So if we wanted it to do it elementwise, we could. We could take the partial derivative with respect to A11, partial derivatives with respect to A12, partial derivative with respect to A21, partial derivative of A22 and so forth. Of course, this is really becomes really awkward.

But this is for matrix value function. But this is exactly equivalent to saying that df is this inner product, grad f dot dA, which is equal to the trace of grad f transposed dA because our inner product really is the elementwise product of the components. And so this gradient is what you expect a matrix gradient to be. But as I said, it's just really annoying to work with in this form.

So maybe I'll do one more. So maybe I'll mention next time, or soon, maybe not next time. I don't know. Soon, we'll also do the gradient of the determinant of A for an m by m matrix. And it turns out this is not so easy to derive. And you definitely would not want to do this component by component. That's a really awkward way to do this. It turns out this is exactly equal to determinant of A times A inverse transposed.

And let me do a simpler example. It also illustrates a nice property of a trace. Let's do f-- and then we'll take a break. So let's do f of A is x transpose A y. So this A is going to be m by n. And this is for some constant x in R something and y in R something. So how many components does x have? Alex, are people shouting out anything. I can't hear anything.

**AUDIENCE:** No. No one said anything.

**STEVEN JOHNSON:** This, I wanted to give a number. Just think in the shapes. This is n by n. Just for this product to even makes sense, how many components does x have to have?

**AUDIENCE:** M?

**STEVEN JOHNSON:** M. And how many components does y have?

**AUDIENCE:** N.

**STEVEN JOHNSON:** N, yes. So if we just take a matrix and sandwich it in between two vectors of the right-- but they have to match the number of rows, the number of columns, then you get a scalar out. And so then what's df? This one is even easier. So df is just x transpose. Everything is a constant. So I'm just going to use the-- you can go to the product rule. But there's only one. So technically, this is the product rule. But dx is 0. dy is 0. Those are constant. So this is just this.

But this is a function that takes in a vector, in this case, a vector space, a matrix, and gives you a scalar. So I should be able to express this as a dot product. So I want to express this as a trace. So I'm going to expresses this somehow as grad f dot dA. So how can I do that? So there's a couple tricks.

The first trick is this is a number. df, this is a scalar. A scalar, I can always write as the trace of itself. The trace of a scalar is just itself if you think of a scalar as a 1 by 1 matrix. So I can always put a trace there. But this is not a trace in the form that I want because I want the dA over on the right.

So let me just put my little notes on the side. Note, if you take the trace of AB-- do people remember this formula? This is this is the same thing as the trace of BA. This is called the cyclic property of the trace. And if you don't remember it, it's pretty easy to convince yourself of it.

So that does not mean I can rearrange this in any order I want. It means if I have two things, I can swap them. So it's called cyclic because if I think of this thing as the first thing and this as the second, I can swap it. I can move y over to there, or I could move the x over to the other side, I can do what's called a cyclic permutation of these things right. So I have to think of them-- to apply this, I have to group this into two pieces.

So I could group this into, for example, x transpose and this. And then I should be able to swap them. And y-- sorry. I don't want to do it in that order, though. I want to swap it in the other way. So this is my cyclic permutation because I want to put the dA in the right. So this is the trace of y x transpose times dA.

Now, this looks like a dot product. So this is-- remember, the dot product is always trace of something transpose dA. So this is really, to make it a dot product, I have to transpose the thing. This is xy transpose dot dA. Does everyone see this? Because remember, the trace of the dot product, again, is-- where is it? The dot product is trace A transpose B.

So I have to view this term here as the transpose of that. And so that's our gradient. So this is our gradient of f. And it's whatever we take the dot product with to get dA. So as you see in the homework, basically, sometimes if you have a vector in and a scalar out, you know it has to be a dot product in some form. If you have a dot product in your vector space, you have to be able to write it in that form.

But sometimes when you apply the derivative rule, it doesn't look like a dot product. You need to do a little bit of rearrangement to get it in the form where it looks like a dot product to get the gradient out. You need to learn a few rules. And they are always of this kind of form, where you need to be able to transpose things or learn which things can be permuted and so forth.

But it's really useful once you get used to this. I could have done the same thing by writing this out in terms of all the components of A and then taking the partial derivative with respect to each component. If you work hard enough, you'll realize that all these partial derivatives, in fact, give you the matrix xy transpose. But it's a lot more work.

This is suddenly now we can take gradients of matrix functions. And in fact, we can take gradients of any scalar function in any vector spaces with a dot product. So maybe later on, we'll talk about even other kinds of vector spaces, where you can define gradients that are even weirder looking than gradients of matrix functions. And in a subsequent lecture, Alan has a nice trick to derive this property of the gradient of a determinant.

**GUEST SPEAKER:** I forgot that I had that. Thanks for the reminder.

**STEVEN JOHNSON:** Yes. But any questions? We'll take maybe a three-minute break.

**AUDIENCE:** No questions.

**STEVEN JOHNSON:** No questions? So let's take a three-minute break. It's 12:11. We'll get back to 12:14.