Matrix differentials

So far we've learned a lot of tools for working with linear functions and linear equations. Of course, in reality, we need to deal with nonlinear functions. One good way to do this is to use linear approximations: work with a linear function that is close to our true nonlinear function in a local region. We can get these linear approximations by differentiating to form a Taylor series; that's the main subject of this set of lecture notes.

A common complication in machine learning is that our nonlinear functions have high-dimensional arguments: multiple vectors, matrices, or even tensors. It's possible but unwieldy to differentiate such functions component by component; instead it's often better and faster to work directly in matrix notation. Tools for the latter are called *matrix differential calculus*.

In these notes we'll use concrete coordinates for all of the vectors and matrices. Many of the same techniques work for abstract vector spaces, but with concrete representations, we get to see how the overall derivatives depend on the derivatives for individual coordinates.

First-order Taylor approximation

We can approximate any nonlinear function $f \in \mathbb{R}^n \to \mathbb{R}^m$ around a point $(\hat{x}, f(\hat{x}))$ with a linear function called the *first-order Taylor approximation* or *Taylor series* for f at $\hat{x}$:

$$y - \hat{y} \approx A(x - \hat{x}) \qquad y, \hat{y} \in \mathbb{R}^m,\ x, \hat{x} \in \mathbb{R}^n,\ A \in \mathbb{R}^{m \times n}$$

Here $y = f(x)$ and $\hat{y} = f(\hat{x})$. And, the matrix A is the *Jacobian* of f at $\hat{x}$; that is, it is the first derivative of y with respect to x, evaluated at $\hat{x}$:

$$A_{ij} = \left.\frac{\partial y_i}{\partial x_j}\right|_{\hat{x}}$$

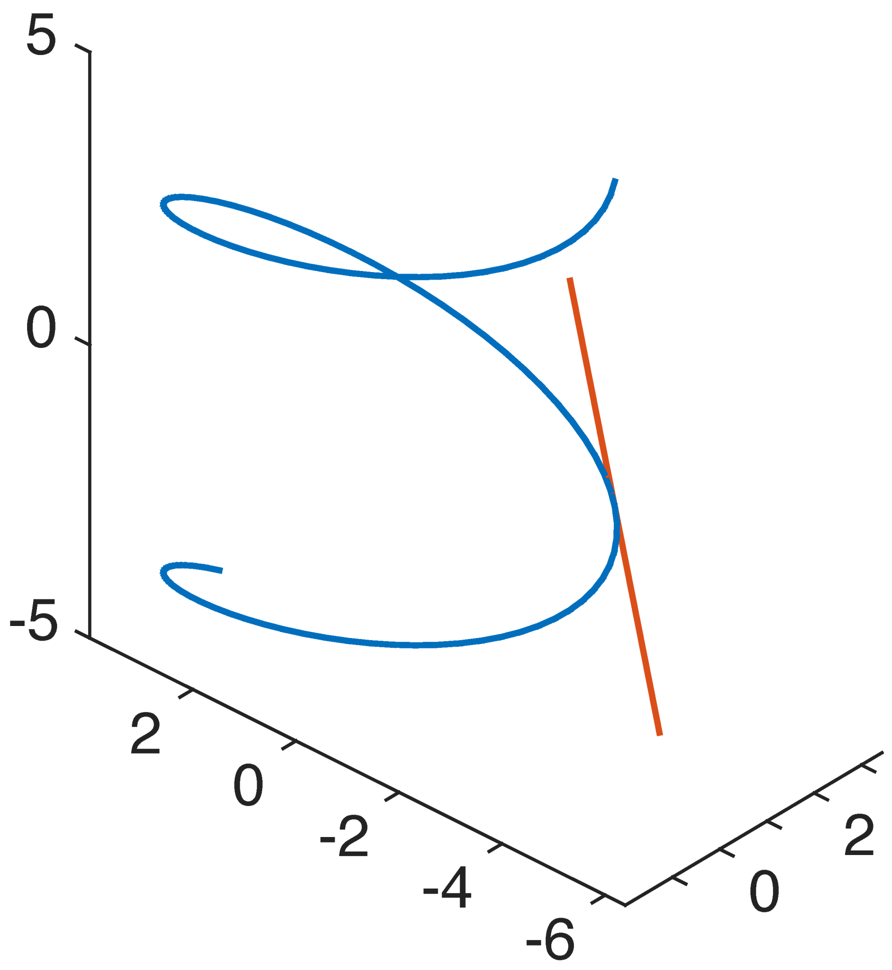
The Jacobian A implicitly depends on the point $\hat{x}$ where we evaluate the derivative. We can be more precise by writing the Taylor approximation as

$$(y - \hat{y}) \approx A(\hat{x})\,(x - \hat{x})$$

Note that there are competing conventions for how to represent the Jacobian: some books define it to be the transpose of what we've written here. Our convention is called the numerator layout, since the numerator (∂y_i) corresponds to the first dimension of A (the row index).

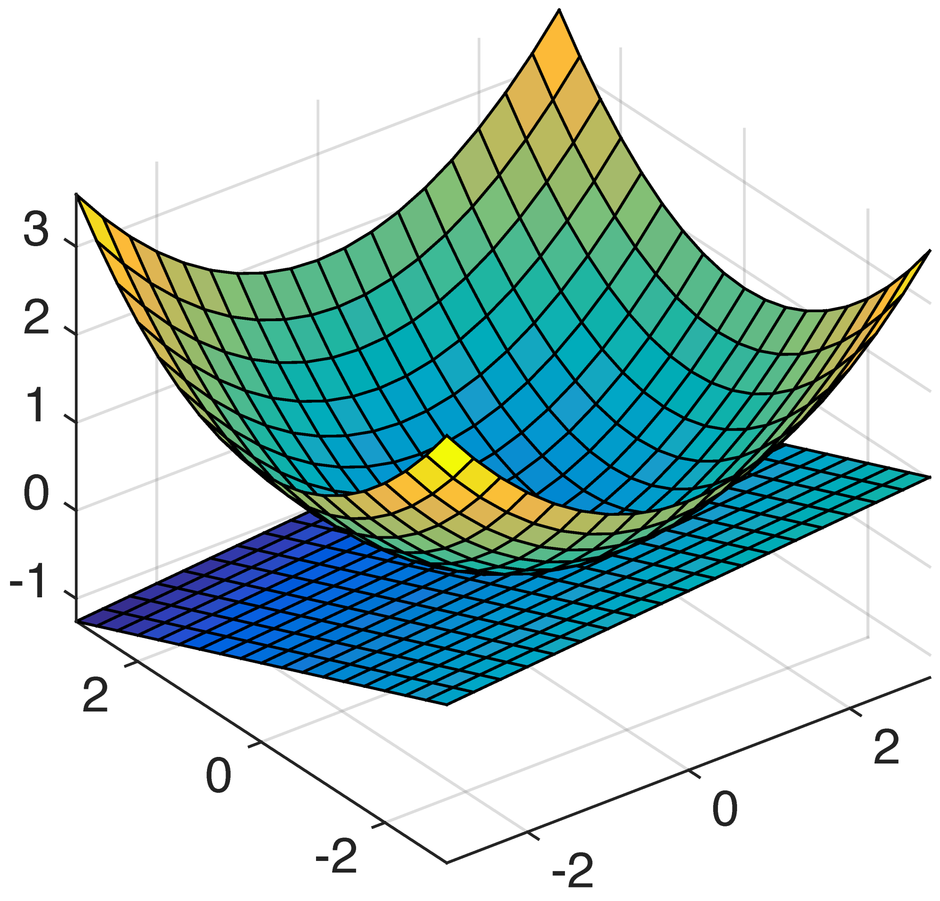
Taylor series examples

If we pick $n = 1$ so that f represents a curve, then the Taylor approximation is a line that is tangent to the curve:



The matrix A will be a column vector, and represents the instantaneous velocity of a point moving along the curve.

If instead we pick $m = 1$, so that f represents a curved surface in $\mathbb{R}^{n+1}$, then the Taylor approximation is a hyperplane (an n-dimensional subspace of $\mathbb{R}^{n+1}$) that is tangent to the surface:



The Jacobian matrix A will be a row vector, equal to the gradient of f. So, the components of A will be the slopes of the hyperplane in each direction.

Differentials

We can also write the first-order Taylor approximation using a somewhat different but equivalent notation:

$$dy = A\,dx$$

Informally, we can think of dy as being short for $y - \hat{y}$, and dx as being short for $x - \hat{x}$. More formally, dy is a vector in the *tangent space* to f at $\hat{x}$, as in the plots in the previous section. (That's why we can write $=$ instead of $\approx$ above: the tangent space is linear even if f is not.)

The vectors dx and dy are called *differentials*. We often think of them as small changes in x and y, and that's a reasonable intuition since the Taylor approximation is typically best in a small region about $\hat{x}, \hat{y}$. But in fact it's perfectly valid to consider large values of dx, dy; it's just that the tangent space may no longer be close to f in this case.

No matter what inner product space we're working in, a differential always has the same type as its original variable. For example, we could have a matrix $A \in \mathbb{R}^{3\times 2}$ that depends on a scalar $x \in \mathbb{R}$. In this case $dx \in \mathbb{R}$ would be a scalar, and $dA \in \mathbb{R}^{3\times 2}$ would be a matrix.

A good mnemonic for the new notation is that if we "divide through by dx", we get $\frac{dy}{dx} = A$, which states that A is the first derivative of y with respect to x. Just like the menmonic for the chain rule, the notation is only suggestive, since we can't actually divide by a vector like dx.

Just as before, we can make it explicit that A depends on x:

$$dy = A(x)\,dx$$

Unlike the notation for the Taylor approximation, note that we've written $A(x)$ instead of $A(\hat{x})$. This convention is traditional, and saves a bit of ink, but it can be confusing: in $A(x)\,dx$, the x in $A(x)$ means the place where we evaluate the derivative, while the x in dx means the variable we are taking the derivative of. There's no ambiguity since one is inside a differential and one is not.

We can use the same sort of notation to work with more-complicated expressions as well. In general, $d(\text{expression})$ means the linear part of a change in the value of expression. So, if we work out a formula for $d(\text{expression})$, it lets us directly read off a first-order Taylor series. For example, the equation

$$d(x^2 + 2x + 3) = 2x\,dx + 2dx$$

says that, if we make a change dx to the value of x, the value of $(x^2 + 2x + 3)$ changes by $(2x + 2)dx$ to first order. (We haven't yet shown how to derive this equation using differential notation, but we can check it using our knowledge of scalar derivatives.)

So why did we bother? The old notation was fine for many purposes, but it turns out that the new notation will be a lot more convenient when we have to manipulate complex expressions. In fact, we will see cases later where it's easy to use the new notation to take derivatives, but it's not even possible to write the answer compactly in the old notation. One reason for the extra flexibility is that we have separated individual differentials like dx and dy, giving each one its own meaning, instead of giving a meaning only to a combined notation like $\frac{dy}{dx}$.

Working with differentials

One of the advantages of differential notation is that it lets us write derivatives of complicated expressions cleanly and compactly. For example, we can mix scalar, vector, and matrix variables; we can avoid explicit manipulation of indices for vectors, matrices, and tensors; and we can work with high-dimensional derivatives without having to reshape them into standard shapes.

Just as with scalar derivatives, some of the most useful tools for working with differentials are linearity, the product rule, and the chain rule. Using these tools, we can often reduce a complex differential to a bunch of lookups of derivatives of standard functions. We'll look at each of these techniques in turn.

Standard functions

For scalar functions like $\cos$ and $\exp$, the differential is equivalent to the scalar derivative:

$$df(x) = f'(x)\,dx$$

So for example, $d\cos x = \sin x\,dx$, $d\exp x = \exp x\,dx$, and $dx^k = kx^{k-1}dx$.

In addition, there are identities for standard matrix and vector functions. For example, if A is a symmetric positive definite matrix, the differential of the log-determinant is

$$d\ln\det A = \langle A^{-1}, dA\rangle$$

where $\langle X, Y\rangle$ is the standard matrix inner product $\operatorname{tr}(X^TY)$. Note the similarity to the scalar logarithm: for scalar $x > 0$, $d\ln\det x = d\ln x = x^{-1}dx$.

For another example, if x is a vector, the differential of the vector norm is

$$d||x|| = \frac{x}{||x||}dx \qquad x \neq 0$$

A final example is that the differential of a constant is zero. More specifically, it is the zero vector of matching type: if $x \in V$ is constant then $dx = 0 \in V$.

There are too many identities for standard functions to list here; a good reference is the Matrix Cookbook.

Linearity

The differential symbol d acts like a linear operator. So for example we can rewrite

$$d(af + bg) = a\,df + b\,dg$$

and we can rewrite

$$d(A^T X + 3Y) = A^T dX + 3\,dY$$

Combining linearity with identities for standard functions lets us go a long way: e.g.,

$$d(3x^7 + 5\sin x + \ln x) = 21x^6 dx + 5\cos x\,dx + x^{-1}dx \qquad x > 0$$

The product rule

The differential of a product is

$$d(fg) = df\,g + f\,dg$$

This is true for any *product-like operation*: e.g., scalar multiplication, matrix multiplication, Kronecker product, composition of linear functions, vector convolution, dot product, or componentwise multiplication of vectors. It's true even when the product is non-commutative, as for example with matrix multiplication. (In this case the order of the terms above matters.) And it extends to more than two terms: e.g.,

$$d(fgh) = df\,gh + f\,dg\,h + fg\,dh$$

In the scalar case the product rule only applies to ordinary multiplication; but for vector spaces there's a nearly infinite variety of product-like operations. For this reason, the generalized version of the product rule can be highly expressive and useful.

Prime notation

The scalar equality $df(x) = f'(x)dx$ is so useful that it's worth upgrading it to work more generally. It turns out that, whenever we have an equation like

$$df(x) = \text{expression involving } x \text{ and } dx$$

we can always factor out dx so that the equation takes the form

$$df(x) = [\text{linear operator that depends on } x]\,dx$$

The overall expression can be nonlinear, since it can have a nonlinear dependence on x. But the differential dx always appears linearly.

For example, if $x = (u, v)^T \in \mathbb{R}^2$, and if $f(x) = u^3 + v^3$, then we can use the rules above (namely linearity of d and the identity for monomials) to show that

$$df(x) = 3u^2\,du + 3v^2\,dv = (3u^2\;3v^2)\,dx$$

As claimed, we have a linear operator (multiplication by the vector $(3u^2, 3v^2)$) applied to $dx = (du, dv)^T$. But the dependence on $x = (u, v)^T$ is nonlinear.

This factorization holds no matter how complicated the type of f is: f could take a tuple of three matrices and a vector as input, and produce a tensor as output, and we would still be able to define a linear function that relates dx to df. But depending on the types involved, the linear function could take different forms: a dot product, a matrix multiplication, or something else. We'll say more about the "something else" case below.

Given the above factorization, it makes sense to give a name to the linear function: we will call it $f'(x)$, so that the equation becomes

$$df(x) = f'(x)\,dx$$

just as for the scalar case. So for example, if $f(u, v) = u^3 + v^3$ as above, then $f'(u, v) = (3u^2, 3v^2)$.

You will sometimes find different conventions for f', such as transposing it or using special notation in some cases for specific kinds of linear operators. The convention we define here has the advantage of uniformity: the equation $df(x) = f'(x)dx$ always holds, and is always interpreted as application of a linear operator to dx.

The chain rule

Now that we've defined prime notation, the chain rule for differentials looks a lot like the chain rule we're used to for scalars:

$$d(f(g(x))) = f'(g(x))\,d(g(x))$$

The easiest way to apply the chain rule is often to write out separate equations for df in terms of dg and for dg in terms of dx. Then we can substitute to get an equation for df in terms of dx.

For example, define a nonlinear function with a matrix argument like

$$f(U) = \ln\det U \qquad g(X) = A^T X A$$

where X and U are symmetric positive definite matrices.

We can use the chain rule to get an expression for $d(f(g(X))$. The first step is to separately differentiate f and g. For f, the identity for $\ln\det$ tells us that

$$df(U) = \langle U^{-1}, dU \rangle$$

making $f'(U)$ the linear operator that maps dU to $\langle U^{-1}, dU \rangle$.

For g, we can use the linearity of d (or the product rule) to show

$$dg(X) = A^T dX\, A$$

Putting these together, we can substitute $U = g(X)$ and apply the linear operator $f'(U)$ to $dg(X)$ to get

$$df(g(X)) = \langle (A^T X A)^{-1}, A^T dX\, A \rangle$$

Example: linear regression

Linear regression is a good example of how to use the tools we've defined above. Recall that, in linear regression, we are given a bunch of data points (x_t, y_t) for $t \in 1 : T$. Here x_t is in a vector space like $\mathbb{R}^d$, and $y_t \in \mathbb{R}$ is a scalar target value. We can collect all of these data points into a single big matrix and vector:

$$X = \begin{pmatrix} | & | & & | \\ x_1 & x_2 & \dots & x_T \\ | & | & & | \end{pmatrix}$$

$$y = \begin{pmatrix} y_1 & y_2 & \dots & y_T \end{pmatrix}$$

The dimensions are $X \in \mathbb{R}^{d \times T}$ and $y \in \mathbb{R}^{1 \times T}$. We'll assume that the inputs x_t already include the effect of any desired feature transform, including adding a constant coordinate if we want one. So, our prediction on the tth data point is $\hat{y}_t = w \cdot x_t$, where $w \in \mathbb{R}^d$ is our parameter vector.

We can collect our predictions for all data points into a vector $\hat{y} \in \mathbb{R}^{1 \times T}$ whose coordinates are $\hat{y}_t$:

$$\hat{y} = w^T X$$

Our prediction errors are then $\varepsilon = y - \hat{y}$, and our sum of squared errors is $\|\varepsilon\|^2$. Our goal is to learn a parameter vector w that minimizes the sum of squared errors:

$$\arg\min_w \varepsilon \cdot \varepsilon = \arg\min_w (y - \hat{y})(y - \hat{y})^T$$

We can minimize by differentiating and setting to zero:

$$0 = d(y - \hat{y})(y - \hat{y})^T = d(yy^T - 2\hat{y}y^T + \hat{y}\hat{y}^T)$$

Since y is a constant, $d(yy^T) = 0$. By linearity and the product rule, and the fact that a scalar is its own transpose,

$$d(-2\hat{y}y^T) = -2d\hat{y}\, y^T$$

$$d(\hat{y}\hat{y}^T) = \hat{y}d\hat{y}^T + d\hat{y}\,\hat{y}^T = 2d\hat{y}\,\hat{y}^T$$

Substituting in and dividing by 2, we now want to solve

$$0 = -d\hat{y}\, y^T + d\hat{y}\,\hat{y}^T = d\hat{y}(\hat{y}^T - y^T)$$

By linearity, $d\hat{y} = dw^T X$, so

$$0 = dw^T X(\hat{y}^T - y^T) = dw^T X(X^T w - y^T)$$

Since dw can be arbitrary, this implies

$$XX^T w = Xy^T$$

which are the normal equations that we introduced earlier.

Exercise: for ridge regression we would add a multiple of the squared norm of w, and solve $\arg\min_w [\varepsilon \cdot \varepsilon + \lambda w \cdot w]$ instead. Differentiate the new objective including the squared-norm term, and find the modified normal equations for ridge regression.

With two or more variables

If we have multiple variables $x, y, \ldots$ then we can collect them together into a tuple u, so that an expression like $f(x, y, \ldots) = \ldots$ becomes $f(u) = \ldots$. Taking the differential, we have

$$df(u) = f'(u)\, du$$

We can then expand back to use the original variables $x, y, \ldots$. Focusing on the two-variable case for simplicity, we get

$$df(x, y) = f_x(x, y)\, dx + f_y(x, y)\, dy$$

That is, we can split the linear operator $f'(u)$ into two separate linear operators $f_x(u)$ and $f_y(u)$, one acting on dx only and one on dy only. Each of these functions represents the partial derivative of $f(x, y)$ with respect to one of its arguments. Just as in the single-variable case, we can *define* the functions f_x and f_y by the above equation.

There are two sets of variables here: the first set $x, y, \ldots$ is the place where we are evaluating the derivative, and the second set $dx, dy, \ldots$ is the change we're making in the first set. The overall expression can be nonlinear in the first set of arguments, but is always linear in the second set.

For example, if $f = \sin(x + y)$ then

$$df(x, y) = \cos(x + y)(dx + dy)$$

by the chain rule. This expression represents the Taylor approximation

$$f(x, y) - f(\hat{x}, \hat{y}) \approx \cos(x + y)((x - \hat{x}) + (y - \hat{y}))$$

Example: two-layer network

Neural networks provide a great example of the chain rule. We can write a two-layer network as

$$y = f(W_2\, g(W_1 x + b_1) + b_2)$$

Here x is the input to the neural network, y is the output, W_1, W_2 are weight matrices, and b_1, b_2 are weight vectors. The functions f and g are transfer functions; often they are componentwise nonlinearities like

$$f_i(z_i) = \ln(1 + \exp(z_i))$$

This particular example is a smooth version of a ReLU; its differential is

$$[df(z)]_i = \frac{\exp(z_i)}{1 + \exp(z_i)}\, dz_i$$

> *In general, whenver we differentiate a componentwise function like this, $df = f'(z)dz$, we can write $f'(z)$ a couple of ways. One is to make a diagonal matrix whose entries are the componentwise derivatives: $[f'(z)]_{ii} = \frac{d}{dz_i} f_i(z)$. Another is to make a vector of componentwise derivatives and use the $\circ$ multiplication-like operator we defined above: $s_i = \frac{d}{dz_i} f_i(z),\ df(z) = s \circ dz$.*

For brevity, write u_1, u_2 for the inputs to the first and second layers of the network:

$$u_1 = W_1 x + b_1 \qquad u_2 = W_2\, g(u_1) + b_2$$

We can now differentiate using the chain rule and linearity, to find how the output y of the network varies as we change the input x:

$$\begin{aligned} y &= f(u_2) \\ dy &= f'(u_2)\, du_2 \\ &= f'(u_2)[W_2\, dg(u_1)] \\ &= f'(u_2)[W_2\, g'(u_1) du_1] \\ &= f'(u_2)[W_2\, g'(u_1)(W_1 dx)] \end{aligned}$$

The backpropagation algorithm for computing gradients is effectively an application of the chain rule, very similar to the above: in backpropagation we want dy as a function of dW_1, db_1, dW_2, db_2, considering x as constant.

A note on notation

When we have multiple variables of different types, expressions for differentials can get complicated, and it may not be straightforward to define a clear notation. One place where notation sometimes fails us is in representing an equation of the form $dy = f'(x)dx$. The value of $f'(x)$ is a linear function, so we have to have notation that lets us apply a linear function of the correct type.

If x and y have simple types, there is often a standard notation for linear function application. E.g., if x and y are real vectors, then $f'(x)$ returns a matrix, and we write linear function application as matrix multiplication.

On the other hand, suppose that our variables are matrices. In this case the output of f' is a linear function between matrices — often represented as a fourth-order tensor. There may be no simple expression for this tensor, even if the calculations involving differentials are not too bad. We'll give an example of this situation shortly.

In the worst case, we can always re-introduce explicit indices and work component by component. To ease index manipulation, we can sometimes use the Einstein summation convention; see `torch.einsum` for more details.

Example: batch norm

For example, consider the *batch normalization* operation that is often used in deep networks. There are two steps in batch normalization: first subtract the batch mean from every example in the batch, and second divide each example in the batch by the batch standard deviation. If we let the matrix X represent our batch, with one column per example in the batch, then we can write these steps as

$$Y = X - \mu(X)e^T$$

$$Z = (V(Y))^{-\frac{1}{2}} \circ Y$$

where e is a vector with all elements equal to 1, the functions μ and V compute row-wise mean and variance, and $\circ$ denotes the broadcasting product. The inverse square root of V is taken coordinatewise. If there are n vectors in our batch, then X will have n columns, and e will be in $\mathbb{R}^n$. We can expand the mean and variance as

$$\mu(X) = \frac{1}{n} X e$$

$$V(Y) = \frac{1}{n} \operatorname{diag}(Y Y^T)$$

where diag extracts the diagonal of a matrix as a vector. (In the definition of $V(Y)$ above, we've simplified the expression by using the fact that the rows of $\mu(X)$ are guaranteed to have zero mean.)

If we want to find dZ in terms of dY, we can first separately differentiate the individual steps in computing Z:

$$dV(Y) = \frac{1}{n} \operatorname{diag}(Y\, dY^T + dY\, Y^T)$$

$$d(v^{-\frac{1}{2}}) = -\frac{1}{2} v^{-\frac{3}{2}} \circ dv$$

$$dZ = \left(-\frac{1}{2} V(Y)^{-\frac{3}{2}} \circ dV(Y)\right) \circ Y + V(Y)^{-\frac{1}{2}} \circ dY$$

The first equation above uses linearity and the product rule. The second equation handles each coordinate separately with the identity for monomials, and puts the results together coordinatewise using $\circ$. The third uses the chain rule (with $dv^{-\frac{1}{2}}$) and the product rule.

Finally we can combine dZ and dV with the chain rule to get the final answer:

$$dZ = -\frac{1}{2n} V(Y)^{-\frac{3}{2}} \circ \operatorname{diag}(Y\, dY^T + dY\, Y^T) \circ Y + V(Y)^{-\frac{1}{2}} \circ dY$$

This does in fact represent a linear function of dY for each value of Y; but there's not an easy way to write this linear function without an explicit representation of its argument dY. We'd have to construct a fourth-order tensor such that an appropriate tensor contraction accomplishes the same thing as the above equation, which is somewhat tricky and error-prone.

On the other hand it's easy to continue working with the above equation. For example, we can calculate

$$dY = dX - d\mu(X)e^T = dX - \frac{1}{n}dX\,ee^T$$

and substitute in to get dZ in terms of dX.

Type checking

We advised earlier that it's generally a good idea to type-check your expressions to help catch bugs: that is, make sure you know the type of each sub-expression, and that every operation makes sense given the types of its arguments. Type-checking is particularly important with differentials, since there can be a lot of different vector spaces floating around.

For this purpose, it helps to be clear about the exact types of expressions involving differentials. Suppose we start from an expression that looks like

$$f = \ldots x \ldots$$

This equation describes how a variable $f \in V$ depends on another variable $x \in U$; in other words it defines a function in $U \to V$. We often call this function f or $f(x)$.

A source of confusion here is the distinction between the function itself (an element of $U \to V$) and its value (an element of V). You will unfortunately see both f and $f(x)$ used to refer to both of these quantities. When working with differential notation, it's probably easiest to use the convention that we omit all placeholder arguments in equations, and assume that all symbols refer to values: e.g., $f = x^2 + 3y$ and not $f(x, y) = x^2 + 3y$. But we'll write $f(x, y)$ if we want to make the arguments of f clear.

The differential will then look like

$$df = f'(x)\,dx$$

where $x \in U$, $dx \in U$, and $df \in V$. Here, $f'(x)$ typically won't appear verbatim; instead we'll get an expression that shows how to apply the linear function $f'(x)$ to dx, like $\langle (A^T X A)^{-1}, A^T dX\, A\rangle$ in the example above.

The type of f' is $U \to (U \to V)$: given x it produces a function that maps $dx \in U$ to $df \in V$. The function f' can be (and often is) nonlinear in its argument x. But the output of f' has to be a linear function that acts on dx; for example, if $U = \mathbb{R}^n$ and $V = \mathbb{R}^m$, then

f' returns a matrix $f'(x) \in \mathbb{R}^{m \times n}$.

Total vs. partial derivatives

One advantage of our new notation is that there's no need for separate treatment of total and partial derivatives ($\frac{d}{dx}$ vs. $\frac{\partial}{\partial x}$). Instead we can use a more flexible and expressive convention: all variables that appear inside a differential can change simultaneously and independently, and any other variables are held constant. (With the $\frac{d}{dx}$ vs. $\frac{\partial}{\partial x}$ convention, it's hard to keep track of mixtures of variables that are either changing together or held constant.) So for example in the expression

$$dL = f(x, y, z)\, dx + g(x, y, z)\, dy$$

we are relating changes in L to changes in x and y while holding z fixed.

If x and y can't change independently, but instead depend on another variable t, we can include more equations:

$$x = \ell(t) \qquad y = m(t)$$

We can separately calculate differentials for each of these new equations:

$$dx = \ell'(t)dt \qquad dy = m'(t)dt$$

and substitute them in if desired:

$$dL = f(\ell(t), m(t), z)\, \ell'(t)dt + g(\ell(t), m(t), z)\, m'(t)dt$$

This substitution is essentially the chain rule. By adding constraints like this, we can implement any combination of variables that change independently or together, or are held constant.

If we want to go back to the old notation, we can manipulate to get an equation that only contains one differential on each side, and then "divide through" by one of them. E.g., if we do this with dt in the equation above, we get the total derivative

$$\frac{dL}{dt} = f(\ell(t), m(t), z)\, \ell'(t) + g(\ell(t), m(t), z)\, m'(t)$$

As always, we aren't really dividing here, so the heuristic doesn't always work; but it's a good mnemonic.

> *What we're really doing is pulling out an expression for f', the linear function that acts on dt. The*

heuristic takes advantage of the fact that we often write application of a linear function like multiplication, but it fails when we can't do this.

Useful identities

Linear operations

We already know that the differential d passes straight through linear operations. Here are a few useful examples:

- Matrix transpose is linear, as is its cousin the adjoint of a linear operator. So, $d(A^T) = dA^T$.
- Matrix trace is linear, so $d\,\mathrm{tr}(A) = \mathrm{tr}(dA)$. The trace shows up in a lot of places; see below.
- Reshaping a vector, matrix, or tensor is linear. So is concatenating or splitting matrices, vectors, or tensors. Common examples include
  - The `numpy.reshape` function.
  - The function that pulls out the main diagonal of a matrix as a vector, $d_i = M_{ii}$, as well as its inverse that turns a vector into a diagonal matrix. Both of these are often written as diag, with the type of the argument determining which one we mean. (So, $\mathrm{diag}(\mathrm{diag}(M))$ returns a matrix that is the same as M on the diagonal, and zero elsewhere.)
  - The function that turns a matrix into a vector by stacking its columns (`M(:)` in Matlab, or `numpy.ravel` in Python)
  - The inverse of the previous example, the function that turns a vector back into a matrix by splitting it into columns and stacking them horizontally.

Trace

The trace of a square matrix $M \in \mathbb{R}^{n\times n}$ is defined as the sum of its diagonal entries,

$$\mathrm{tr}(M) = \sum_{i=1}^{n} M_{ii}$$

As mentioned above (and as can be seen from the definition), tr is a linear function. The trace often finds its way into matrix algebra for a number of reasons. Perhaps the most common one is that the matrix inner product can be expressed using trace:

$$\langle A, B \rangle = \operatorname{tr}(A^T B) = \sum_{ij} A_{ij} B_{ij}$$

Some useful identities involving trace:

- If a is a scalar, we can treat it as a 1×1 matrix, so that $\operatorname{tr}(a) = a$.
- The trace of a matrix is the same as the trace of its transpose, $\operatorname{tr}(A) = \operatorname{tr}(A^T)$.
- We can do *trace rotation*: $\operatorname{tr}(AB) = \operatorname{tr}(BA)$. Trace rotation works as long as the product AB is square, even if A and B are not. In this case, BA is also square, but will be a different size than AB.
- Componentwise multiplication can re-associate: $\operatorname{tr}((A \circ C)^T B) = \operatorname{tr}(A^T (B \circ C))$. It works with broadcasting too, though we have to be careful about keeping the order the same. (This identity works because a diagonal operator is self-adjoint.)

Exercise: prove that trace rotation works by expanding the matrix products AB and BA as summations. Prove the identity about componentwise multiplication the same way.

Inverses and determinants

We can get a lot of power from formulas for the differentials of common matrix operations such as inverse and determinant. Let F be an $n \times n$ matrix. If F is invertible, then

$$d(F^{-1}) = -F^{-1} dF \, F^{-1}$$

When F is not invertible, the above identity fails: dF doesn't exist, similar to what happens if we try to differentiate $1/x$ at 0.

Similarly, if F is invertible,

$$d \det F = (\det F) \operatorname{tr}(F^{-1} dF)$$

When F is not invertible, the above identity fails: while the differential still exists, the formula for it is more complex, and is beyond the scope of these notes.

If F is positive definite, then

$$d \ln \det F = \operatorname{tr}(F^{-1} dF)$$

When F is not positive definite, the above identity fails: the slope approaches infinity as we approach the boundary of the set of PSD matrices.

> *For completeness, here's a quick definition of the matrix determinant. For a lower or upper triangular matrix T, $\det T$ is the product of its diagonal elements, $\prod_i T_{ii}$. For a general matrix $A = LU$, the determinant is the product of the determinants of its factors, $\det A = (\det L)(\det U)$. While it's not obvious, the determinant turns out to encode a lot of useful information about a matrix, so it shows up regularly in formulas that we need to differentiate.*

Shapes of derivatives

In the equation $dy = f'(x)dx$, the shape of f' depends on the shapes of x and y. Here's a table of common shapes: write a, s, t for scalars, u, v for column vectors, and M, N for matrices. Then

	Scalar	Vector	Matrix
Scalar	$ds = a\,dt$	$ds = u^T dv$	$ds = \mathrm{tr}(M^T dN)$
Vector	$du = v\,ds$	$du = M\,dv$	$\times$
Matrix	$dM = N\,ds$	$\times$	$\times$

For example, the middle entry in the top row tells us that, if we have a scalar s that depends on a vector v, then $f'(v)$ will be a row vector, written here as u^T. In general, the row label tells us the shape of the output, the column label tells us the shape of the input, and the entry at that row and column tells us the shape of f'.

This table lets us map differentials to more traditional names:

- In the table entry $ds = a\,dt$, the scalar a is the ordinary derivative.
- In the table entry $ds = u^T dv$, the vector v is the gradient of a real-valued function.
- In the table entry $du = v\,ds$, the vector v is the tangent to a curve in $\mathbb{R}^d$.
- In the table entry $du = M\,dv$, the matrix M is the Jacobian of a function from one vector space to another.
- In the remaining entries, there's no traditional name other than "derivative."

Each of the entries of the table is called an *identification theorem*: it lets us identify an expression involving differentials with an expression about ordinary derivatives. For example, in a previous section we showed how to differentiate the log-determinant function:

$$d \ln \det F = \operatorname{tr}(F^{-1} dF)$$

Given this equation, the top-right entry of the table lets us identify the derivative as

$$\frac{d}{dF} \ln \det F = F^{-T}$$

We can summarize all of the identification theorems into the single identity $dy = f'(x)dx$, where we identify $f'(x) = \frac{dy}{dx}$ as the derivative.

This table also shows us something else interesting: in the entries marked $\times$, traditional derivative notation becomes more difficult. While we could write the derivative as a third or higher order tensor, often it's simpler to avoid doing so. An example of this is in the section above on batch norm: while we could have forced our equations into the form $dx = f'(y)dy$ so that we could pull out an expression for the tensor $f'(y)$, it's easier if we don't.

Second and higher differentials

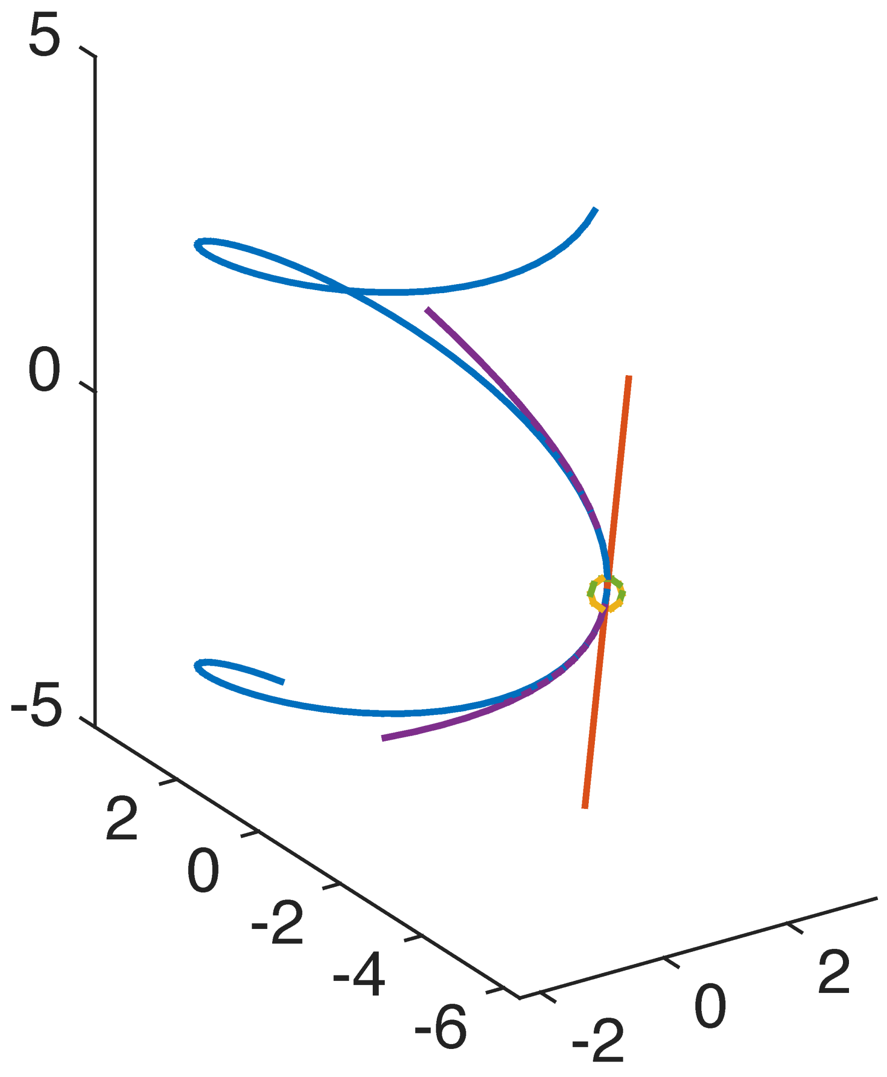
Suppose we've followed the above rules to calculate a differential $df = f'(x)dx$. This equation represents a first-order Taylor expansion, showing how the change df depends on the change dx. This expansion is accurate in some neighborhood of the point x.

We could ask instead for a second-order Taylor approximation — this will typically be accurate in a larger neighborhood of x. We can write the second order Taylor expansion as

$$y \approx f(x) + df(x) + d^2 f(x)$$

where df and $d^2 f$ are the first and second order differentials. (We'll show how to find $d^2 f$ below.) Just as the first differential $df(x)$ represents the linear part of the change in f as a function of the change in x, the second differential $d^2 f(x)$ represents the quadratic part.

Here's an example of a second-order Taylor approximation (in purple) along with the first-order approximation (in red):



The first-order expansion is a *linear* function of dx; the second-order expansion will be a *quadratic* function of dx.

In this figure,

$$f(t) = \begin{pmatrix} 2\cos t \\ 4\sin t \\ t \end{pmatrix}$$

and the differentials are

$$df(t) = f'(t)\,dt = \begin{pmatrix} -2\sin t \\ 4\cos t \\ 1 \end{pmatrix} dt$$

$$d^2 f(t) = f''(t)\, dt^2 = \begin{pmatrix} -2\cos t \\ -4\sin t \\ 0 \end{pmatrix} dt^2$$

Notation for higher differentials

There are only two shapes where standard notation makes it easy to write the quadratic part of a second order Taylor expansion:

- If t is a scalar with $s = f(t)$, then $f''(t)$ is the same shape as s, and $d^2 s = f''(t)\, dt^2$.
- And if s is a scalar and u is a vector with $s = f(u)$, then we can write $f''(u)$ as a matrix, and $d^2 s = du^T f''(u)\, du$.

More generally, for arbitrary shapes x and y, we can write $d^2 y = f''(x)\, dx \otimes dx$. (In fact, we can consider this as a definition of $f''(x)$.) This formula states that the second differential is a linear function (represented by $f''(x)$) applied to the square of dx. The symbol $\otimes$ represents the outer product: the vector that contains all possible pairwise products of components of dx. However, this formula isn't that useful in practice: it's often better not to try to pull out an explicit expression for $f''(x)$.

We can go even higher, to third differentials, fourth differentials, and so on. For the third differential we get a cubic equation that we can write $d^3 y = f'''(x)\, dx \otimes dx \otimes dx$ (a linear function applied to the third power of dx). The fourth differential yields a quartic equation, and so on. But most of the time, it's enough to work with first and maybe second order differentials; and as before, even if we do work with a third order differential, we often don't want to pull out an explicit expression for $f'''(x)$.

Computing the second differential

Computing the second differential works exactly like computing the first differential. We need just one extra rule: pre-existing copies of dx behave like constants.

For example, start by finding the first differential of $\frac{1}{2}\|Ax - b\|^2$:

$$\begin{aligned} \|Ax - b\|^2 &= (Ax - b)^T (Ax - b) \\ d\,\tfrac{1}{2}\|Ax - b\|^2 &= \tfrac{1}{2} d(x^T A^T A x - 2b^T A x + b^T b) \\ &= \tfrac{1}{2}(dx^T A^T A x + x^T A^T A\, dx - 2b^T A\, dx) \\ &= x^T A^T A\, dx - 2b^T A\, dx \end{aligned}$$

From the above, we can find the second differential:

$$\begin{aligned} d^2\,\tfrac{1}{2}\|Ax-b\|^2 &= d(x^T A^T A dx - 2b^T A dx) \\ &= dx^T A^T A\, dx \end{aligned}$$

The entire second term vanishes because it is constant; in the first term we use linearity to pull out the constant $A^T A\, dx$.

Multiple variables

The same rule applies when we have multiple variables: we consider all previous instances of differentials $dx, dy, \ldots$ to be constant. For example, if

$$f(x,y) = (\cos x)(\sin y)$$

then

$$df = (-\sin x\, dx)\sin y + (\cos x)(\cos y\, dy)$$

by the product rule; so

$$\begin{aligned} d^2 f = (-\cos x\, dx^2)\sin y + (-\sin x\, dx)(\cos y\, dy) \\ + (-\sin x\, dx)(\cos y\, dy) + (\cos x)(-\sin y\, dy^2) \end{aligned}$$

The first two terms in $d^2 f$ come from the first term of df by the product rule; the last two terms of $d^2 f$ come from the last term of df by the product rule.