Multivariate Calculus I: Derivatives and local geometry

Nathan Ratliff

Nov 22, 2014

Abstract

This document takes a first step beyond the tools of linear algebra and addresses the much larger space of nonlinear functions and maps that play a big role in many areas of machine learning, robotics, and optimization. We introduce the basics of higher-dimensional derivatives, focusing primarily on what they tell us about the local geometry of nonlinear maps and functions. Our journey takes us from Taylor approximations, which reveal geometry through connections back to the familiar tools of linear algebra, to the rote mechanics of derivative calculations. We discuss calculational tricks and tools emphasizing basic principles and post hoc geometric interpretation over formulas. And we explore in-depth the specific problem of calculating the positional Jacobian of a revolute manipulator to demonstrate how returning to the basic definition of what the Jacobian means allows us to side-step the typical cumbersome machinery of differential calculus and enable the direct derivation of a geometrically intuitive and computational effective algorithm for computing what has proven to be a critical mathematical tool in robotics.

1 Onward to nonlinear functions and smooth mappings

Linear algebra offers a powerful tool chest that's seen widespread application. But, alas, not everything is linear. We saw an example of a robot that did admit linear end-effector control in Figure 1 (left). That robot had simple linear hydraulic pistons for joints that simply grew or shrunk the links. It was linear, and we could answer any number of questions about how to get its end-effector to do what we want. But most robots aren't that restrictive. We chose this robot design specifically to illustrate our simple linear analysis; generally, requiring linearity constrains the types of systems we can design and restricts what they can do. Much more often we find robots like that pictured in Figure 1 (right) which consist primarily of revolute (rotational) joints. These robots more naturally model the types of kinematic abilities we see in our own arms and legs. Nature's stumbled on a good idea and we want to use it, so suddenly we find

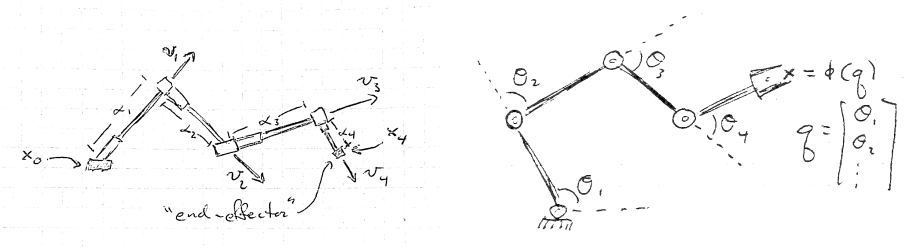

Figure 1: **Left:** Entirely prismatic planar robot using linear actuators which admits direct linear analysis. **Right:** Planar revolute robot with only rotational joints. This robot doesn't immediately lend itself to a linear analysis.

ourselves in the realm of nonlinear smooth mappings and seemingly beyond the reach of linear algebra. A simple trigonometric analysis of the pictured revolute robot tells us that the position of the end-effector is a highly nonlinear function of the joint configuration requiring a whole collection of sines and cosines. So what can we do?

This document takes a first step into the study higher-dimensional derivatives of *nonlinear* functions and mappings. In our fields of interest—machine learning, robotics, optimization, etc.—it's important to be fluent at deriving derivatives. We'll review some tricks-of-the-trade toward this end, including some hints on how to gain intuition for the resulting expressions and how to definitively check that your answer is correct on a computer. But perhaps more importantly we'll study how these derivatives give rise to a geometric understanding of their underlying nonlinear functions and maps. And, more fundamentally, we'll see that derivatives, especially first and second derivatives, are what connect nonlinear functions back to the powerful tools of linear algebra.

This is the stuff of differential multivariate calculus—also called advanced calculus, multidimensional calculus, or vector calculus. We won't be discussing fundamental theorems related to integration such as the Stokes's theorem or the concepts behind flux, curl, and the likes. Those elements are fundamental to the core of multivariate calculus and are especially important for an in depth understanding of physics and related areas, but they tend to be less important for our fields of interest. Instead, we'll focus primarily on calculational techniques and geometry which are a prerequisite to understanding the basic machinery behind fields such as robotics, machine learning, and optimization of central importance to intelligent systems.

2 Building higher-dimensional derivatives from one-dimensional derivatives

Remembering back to univariate (one-dimensional) calculus, we're reminded that derivatives are simply rates of change. How much does the value of a function $f : \mathbb{R} \to \mathbb{R}$ change when we change its input? Naïvely, we can write this expression as a ratio $(f(x + \Delta x) - f(x))/\Delta x$, for some finite Δx (just perturb the input a little and compare it to the size of the perturbation), but, of course, the size of any *finite* perturbation Δx is entirely arbitrary. And, indeed, it increasingly accurately answers our question about the rate of change *specifically* at the point x as the size of that perturbation gets smaller and smaller. So, more definitively, we define this rate of change, or derivative, as the limit of that ratio as Δx becomes increasingly small. Notationally, we write

$$\frac{d}{dx} f(x) = \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}. \tag{1}$$

And if this limit exists everywhere, we say the function is *differentiable*.

But now we have functions with more inputs $h : \mathbb{R}^n \to \mathbb{R}$. What do we do with those? In general, the question is a little complicated to answer, because we don't actually have a function of multiple inputs, we have a function defined on a vector space represented in a particular coordinate system which means we have to talk more generically about taking slices of the function and looking at derivatives in particular directions. If we do that analysis, though, we find that with mild conditions on the function the situation becomes a bit easier. Rather than digging into the drudgery of that analysis (actually, it's quite fascinating, but beside the point for our purposes), we'll typically assume that the functions we're dealing with are "nice" in these ways and proceed with the simpler, more intuitive, setting that results.

2.1 Building the Gradient

Every multi-dimensional function $f(x_1, x_2, \ldots, x_n)$ can be turned into a more familiar one-dimensional function simply by keeping all of the inputs fixed except one $f_k(x_k) = f(x_1, \ldots, x_k, \ldots, x_n)$. Then the derivative of the resulting one-dimensional function is what we call the *partial derivative* of f with respect to x_k. It's denoted

$$\begin{aligned} \frac{\partial}{\partial x_k} f(\boldsymbol{x}) &= \frac{d}{dx_k} f_k(x_k) \\ &= \lim_{\Delta x_k \to 0} \frac{f(x_1, \ldots, x_k + \Delta x_k, \ldots, x_n) - f(x_1, \ldots, x_k, \ldots, x_n)}{\Delta x_k}. \end{aligned} \tag{2}$$

To fully describe the different ways in which the function varies with all of its inputs, we simply need to collect up all these partial derivatives, one for each dimension. The resulting column vector of partial derivatives is called the

gradient, and it's denoted

$$\nabla f(\boldsymbol{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix}. \tag{3}$$

Each component of this vector tells us how the function varies when only that dimension changes.

2.2 Building the Hessian

Section 3 discusses the geometry of this gradient in detail, but before we get there, lets first define one more multi-dimensional derivative quantity that arise quite often in practice. This mathematical object is the multi-dimensional version of the second derivative. Again, we simplify the problem by reducing it to partial derivatives, but now we want to know how the function changes when two of the inputs are perturbed. Notationally, we write $h_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j}$ and collect up all of these elements into a matrix to form what we call the **Hessian**:

$$\nabla^2 f(\boldsymbol{x}) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}. \tag{4}$$

We won't prove it here, but it's easily verified by example that the order of partial derivatives doesn't matter when the second derivative is continuous. Therefore, for such functions

$$h_{ij} = \frac{\partial}{\partial x_i}\left(\frac{\partial f}{\partial x_j}\right) = \frac{\partial}{\partial x_j}\left(\frac{\partial f}{\partial x_i}\right) = h_{ji}, \tag{5}$$

which means the Hessian must be symmetric, i.e. $\nabla^2 f(\boldsymbol{x}) = \nabla^2 f(\boldsymbol{x})^T$.

These two objects, the gradient and the Hessian, play the role of the multi-dimensional versions of first and second derivatives.

2.3 The total derivative of a function

The gradient gives us n numbers which tell us how the function changes with a unit change to each of its inputs. But now suppose we actually change each of the inputs by small (but different) amounts Δx_k. How much does the function change in total? If a unit change to the kth input changes the function's output by $\frac{\partial f}{\partial x_k}$ then the total contribution to the change in output by changing the kth input by Δx_k (as long as Δx_k is small) is the rate of change times the amount of

change: $\frac{\partial f}{\partial x_k}\Delta x_k$. Adding all these contributions together gives a total change of

$$\sum_{k=1}^{n} \frac{\partial f}{\partial x_k}\Delta x_k = \nabla f(\boldsymbol{x})^T \Delta \boldsymbol{x}, \tag{6}$$

where $\Delta \boldsymbol{x}$ is the vector constructed from components Δx_k.

Thus, if the input is actually a function of another variable, say time, so that $\boldsymbol{x} : \mathbb{R} \to \mathbb{R}^n$, we can ask the question how does the function change *in total* with a small change to the time variable. This mapping from the time variable to a point in $\mathbb{R}^n$ forms an n-dimensional *trajectory* through the space. The rate at which that trajectory is changing at any moment in time is given simply by the first derivatives of each of its component functions. We often denote this rate of change, or velocity, as

$$\frac{d\boldsymbol{x}}{dt} = \begin{bmatrix} \frac{dx_1}{dt} \\ \frac{dx_2}{dt} \\ \vdots \\ \frac{dx_n}{dt} \end{bmatrix} = \dot{\boldsymbol{x}}. \tag{7}$$

Typically, as indicated on the right, we denote this first time derivative (the velocity) using a dot over the vector. This is the notation used in most engineering and physics texts.

The velocity gives us a vector that tells us how each dimension is changing at a given moment in time. Equation 6 then tells us that the total rate of change to the function at that moment of time is

$$\frac{d}{dt} f(\boldsymbol{x}(t)) = \sum_{k=1}^{n} \frac{\partial f}{\partial x_k}\frac{dx_k}{dt} = \nabla f(\boldsymbol{x})^T \dot{\boldsymbol{x}}, \tag{8}$$

There are rigorous ways to establish this formula, but this argument introduces some intuition to keep in mind when thinking through the mechanics of how derivatives propagate. This formula is a version of the multi-dimensional Chain Rule which we explore in more detail in Section 4. Generally, the derivative of a complicated set of nested functions with respect to some underlying variable governing the system, such as the time variable here, is called the **total derivative** of the function. Generally, all *partial derivatives* with respect to variables that are in some way connected to the underlying variable come into play when computing the *total derivative*.

3 "Best fit" approximations: Taylor truncated

Remember that in one-dimensional calculus, the derivative, which represents a rate of change, can be used to form a *tangent* to the function (see Figure 2). Let $\sigma : \mathbb{R} \to \mathbb{R}$ be a one-dimensional function and denote its first derivative by

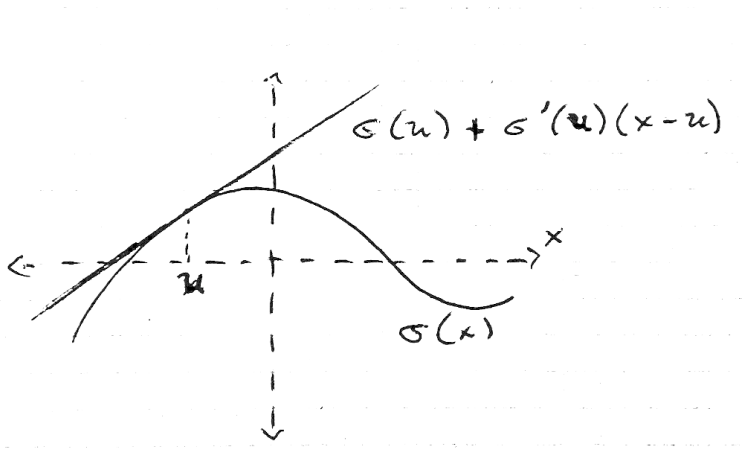

Figure 2: Depicts the tangent to a one-dimensional function.

$\sigma'(u)$ at $u \in \mathbb{R}$. By definition, the derivative is the limit of the slope, how the function changes per unit change of the input. So if we multiply by any change of the input $\Delta x = x - u$, we get an approximation of the value of the function

$$\sigma(x) = \sigma(u + \Delta x) \approx \sigma(u) + \sigma'(u) \underbrace{(x - u)}_{\Delta x}. \tag{9}$$

If the slope $\sigma'(u)$ at u correctly characterizes the slop everywhere (i.e. the derivative is constant), then this approximation is exact. Otherwise, it's best close to u, in a region where the derivative is well approximated by $\sigma'(u)$. Importantly, this approximation acts as a good linear approximation to the function at x, as long as x is sufficiently close to the point around which we took the derivative u.

Since each partial derivative is defined as a regular one-dimensional derivative for the "locked" versions of the function $f_k(x_k)$, we can do the same thing with each individual component of the gradient. Consider the kth such function $f_k(x_k)$. The one-dimensional tangent function along that single dimension is

$$\widetilde{f}_k(x_k) = f(\boldsymbol{x}) + \frac{\partial f}{\partial x_k}(\boldsymbol{x})\epsilon_k, \tag{10}$$

where ϵ_k is some perturbation away from x_k. We can combine all of these one-dimensional tangent functions into a tangent plane simply by taking all weighted averages of them. Suppose $\alpha_1, \ldots, \alpha_n \geq 0$ are non-negative weights that sum to one $\sum_k \alpha_k = 1$. A weighted combination of these one-dimensional tangent

functions is

$$\begin{aligned}\sum_k \alpha_k \widetilde{f}_k(x_k) &= \sum_k \alpha_k \left(f(\boldsymbol{x}) + \frac{\partial f}{\partial x_k}(\boldsymbol{x})\epsilon_k \right) \\ &= f(\boldsymbol{x}) + \sum_k \frac{\partial f}{\partial x_k}(\boldsymbol{x})\alpha_k\epsilon_k \\ &= f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^T \Delta\boldsymbol{x}, \end{aligned} \tag{11}$$

where $\Delta\boldsymbol{x}$ is a vector of perturbations away from $\boldsymbol{x}$ whose components consist of $\alpha_k\epsilon_k$. This function is linear, and its graph represents the plane tangent to the function at $\boldsymbol{x}$.

In other words, we've constructed a "best" affine approximation to the multi-dimensional function from our understanding of one-dimensional calculus. This analysis touches on the fundamental expansion of a function in terms of its derivatives: The multi-dimensional generalization of the Taylor expansion. We explore this idea more in the next section.

3.1 Generalizing Taylor to higher dimensions

Consider again a one-dimensional function $\sigma : \mathbb{R} \to \mathbb{R}$. We remember from one-dimensional calculus that if the function is infinitely differentiable the on an open interval around a point $u \in \mathbb{R}$, we can expand the function in terms of its derivatives as

$$\sigma(x) = \sigma(u) + \sigma'(x-u) + \frac{1}{2}\sigma''(u)(x-u)^2 + \cdots + \frac{1}{k!}\sigma^{(k)}(u)(x-u)^k + \cdots$$

Intuitively, this expression recursively gives us the best constant approximation (the first term), the best linear approximation (the first two terms), the best quadratic approximation (the first three terms), and so on down the line. These kth-order polynomial approximations are the "best" in the sense that their first k derivatives, by construction, match those of the original function at the expansion point u. Each term tells us how the function varies away from u to the kth order.

We've already seen how first terms carry over in Equation 11. Writing it more explicitly using this notation, we get

$$f(\boldsymbol{x}) \approx f(\boldsymbol{u}) + \nabla f(\boldsymbol{u})^T(\boldsymbol{x} - \boldsymbol{u}). \tag{12}$$

This is the **first-order Taylor approximation** of f around $\boldsymbol{u}$. It's the unique "best" fit linear approximation of f whose value and first derivative (the gradient) matches the original function. We can fine-tune the approximation by adding a term representing how the function varies to the *second* order away from $\boldsymbol{u}$ using the matrix of second derivatives (the Hessian):

$$f(\boldsymbol{x}) \approx f(\boldsymbol{u}) + \nabla f(\boldsymbol{u})^T(\boldsymbol{x} - \boldsymbol{u}) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{u})^T\nabla^2 f(\boldsymbol{u})(\boldsymbol{x} - \boldsymbol{u}). \tag{13}$$

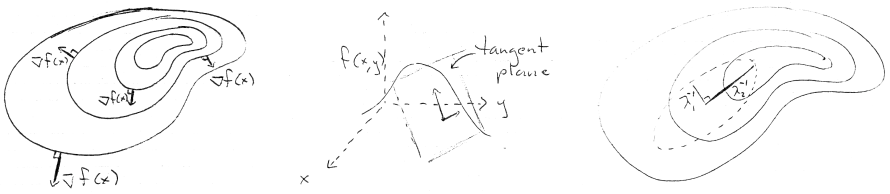

Figure 3: **Left:** The gradient is always orthogonal to the isocontours of the function. **Middle:** The graph of the first-order Taylor approximation is tangent to the graph at the evaluation point. **Right:** The Eigenspectrum of the Hessian relates how the function is shaped around the evaluation point. The figure shows the unit isocontour of the quadratic form $\frac{1}{2}(\boldsymbol{x}-\boldsymbol{u})^T\nabla^2 f(\boldsymbol{u})(\boldsymbol{x}-\boldsymbol{u})$.

We call this extended approximation the **second-order Taylor approximation**. Now it's the unique "best" fit quadratic approximation at $\boldsymbol{u}$ whose value, first partial derivatives (the gradient), *and* second partial derivatives (the Hessian) match the original function.

These approximations are of fundamental importance in a number of fields because they link general smooth nonlinear functions back to things we can manipulate: linear and quadratic functions. These approximations allow us to bring all the linear algebra tools we've been studying to bear on otherwise very difficult analytical and algorithmic problems.

3.2 Geometry of Taylor approximations

The gradient and Hessian, and therefore the first- and second-order Taylor approximations, tell us a lot about the geometry of the function. Specifically, the gradient tells us how the function is slopped at a given point, and the Hessian, through its Eigenspectrum, tells us how the function is shaped or how it curves in various directions. We'll explore these two properties a bit here.

If we imagine these approximations in just two-dimensions, with a graph extending into the third dimension, we can think of the first-order Taylor approximation simply as a tangent plane (we built it by construction this way in Section 3). But notice also that the linear approximation consists primary of an inner product between $\nabla f(\boldsymbol{u})$ and the perturbation direction $\Delta\boldsymbol{x} = \boldsymbol{x} - \boldsymbol{u}$ away from $\boldsymbol{u}$. That inner product is the evaluation of a rank 1 linear map $\boldsymbol{A} = \nabla f(\boldsymbol{u})^T$ that has an entire $n-1$ dimensional null space of vectors orthogonal to the gradient direction that are mapped to 0. In terms of the linear approximation, these vector perturbations away from $\boldsymbol{u}$ form the directions along which the value of the function doesn't change, where it's always just the constant value $f(\boldsymbol{u})$. As we see in Figure 3 (left), this null space of directions orthogonal to the gradient form an $n-1$ dimensional hyperplane *tangent* to the isocontours of value $f(\boldsymbol{u})$. (If not, then for small enough epsilon away from $\boldsymbol{u}$ we'd be able to find a point along the isocontour that changes in value, which is a contradic-

tion.) This analysis shows that **the gradient is always perpendicular to the isocontours of the function!**

This observation tells us that the gradient is the direction of *steepest increase* if we measure angle in terms of simple dot products between coefficient vectors. The gradient is the direction that is entirely orthogonal to the equipotential surface of the function at $\boldsymbol{u}$. Moving in that direction has no inefficiencies in the sense that there's no component of the vector along which the function doesn't increase.

The second geometric property of the first-order Taylor approximation is that its graph, as shown in middle plot of Figure 3, is tangent to the graph of the full function. We've already explored this property in Section 3, so we refer the reader back to that section for more details.

Finally, lets take a look at the quadratic term of the second-order Taylor approximation. The second-order term is simply a zero-centered quadratic function

$$q_f(\boldsymbol{x}) = \frac{1}{2}(\boldsymbol{x} - \boldsymbol{u})^T \nabla^2 f(\boldsymbol{u})(\boldsymbol{x} - \boldsymbol{u}). \tag{14}$$

Since the Hessian $\nabla^2 f(\boldsymbol{u})$ is a symmetric matrix, we can always find an orthogonal basis of Eigenvectors $\mathcal{B} = \{\boldsymbol{e}_i\}_{i=1}^n$ with corresponding real Eigenvalues λ_i. As we've seen before, the Eigenspectrum tells us a lot about the geometry of a multi-dimensional quadratic. In this case, we can think of the zero-centered quadratic simply as the sum of n one-dimensional zero-centered quadratics, each with quadratic coefficient λ_i, and each aligned with Eigenvector direction $\boldsymbol{e}_i$. Specifically, we can rewrite the Hessian in terms of its Eigenvectors and Eigenvalues by diagonalizing it as $\nabla^2 f(\boldsymbol{u}) = \sum_{i=1}^n \lambda_i \boldsymbol{e}_i \boldsymbol{e}_i^T$. Plugging this expansion into Equation 14 gives

$$\begin{aligned} q_f(\boldsymbol{x}) &= \frac{1}{2}(\boldsymbol{x} - \boldsymbol{u})^T \left[\sum_{i=1}^n \lambda_i \boldsymbol{e}_i \boldsymbol{e}_i^T \right] (\boldsymbol{x} - \boldsymbol{u}) \\ &= \sum_{i=1}^n \frac{1}{2} \lambda_i \left(\boldsymbol{e}_i^T (\boldsymbol{x} - \boldsymbol{u}) \right)^2. \end{aligned} \tag{15}$$

Here $\boldsymbol{e}_i^T(\boldsymbol{x} - \boldsymbol{u})$ is just the component of the perturbation $\Delta\boldsymbol{x} = \boldsymbol{x} - \boldsymbol{u}$ along the ith Eigenvector. So this analysis shows that each term is a quadratic function of the component of the perturbation along the ith Eigenvector with coefficient given by the corresponding ith Eigenvalue.

For one-dimensional quadratic functions, we know the coefficient on the quadratic term defines how quickly the function changes as we move away from its minimum, and even whether it's curving upward or downward. In this case, the coefficients of these one-dimensional quadratic functions are the Eigenvalues λ_i. These Eigenvalues define how quickly the function curves along each of the orthogonal Eigenvector directions and whether it's curving up or down along that direction. Note that there's often a mixture of positive and negative Eigenvalues at any given point suggesting that the function there is neither purely

curving upward (convex) or purely curving downward (concave). The rightmost plot of Figure 3 depicts how the Eigenstructure of the Hessian reveals the local curvature of the function.

4 Multi-dimensional maps: Derivatives with multiple outputs

So far we've been discussing only multi-dimensional functions. Gradients and Hessians of multi-dimensional functions collect up the first and second partial derivatives of a function and with those we can create first- and second-order Taylor expansions of the function to locally gain geometric insight that we can leverage to bring to bear our tools from linear algebra.

But what about multi-output maps? Suppose $\boldsymbol{g} : \mathbb{R}^n \to \mathbb{R}^m$ is a smooth map from an n-dimensional space to an m-dimensional space, and denote its output as $\boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{y}$. We can view this vector-valued function as a set of m separate scalar valued functions $g_i : \mathbb{R}^n \to \mathbb{R}$ so that

$$\boldsymbol{g} = \begin{pmatrix} g_1(\boldsymbol{x}) \\ g_2(\boldsymbol{x}) \\ \vdots \\ g_m(\boldsymbol{x}) \end{pmatrix}. \tag{16}$$

Now, for every output function g_i, we have n partial derivatives, one for each input variable $x_1, \ldots, x_n$. Our convention in this setting is to shape the collection of all partial derivatives into an $m \times n$ matrix of the form

$$\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{x}} = \begin{pmatrix} \frac{\partial g_1}{\partial x_1} & \frac{\partial g_1}{\partial x_2} & \cdots & \frac{\partial g_1}{\partial x_n} \\ \frac{\partial g_2}{\partial x_1} & \frac{\partial g_2}{\partial x_2} & \cdots & \frac{\partial g_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial g_m}{\partial x_1} & \frac{\partial g_m}{\partial x_2} & \cdots & \frac{\partial g_m}{\partial x_n} \end{pmatrix} = \begin{pmatrix} \nabla g_1^T \\ \nabla g_2^T \\ \vdots \\ \nabla g_m^T \end{pmatrix}. \tag{17}$$

We call this matrix the Jacobian and denote it using the notation $\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{x}}$. Notice that it's actually the *rows* of this matrix that hold each output's *transposed* gradient. That seems to be a flip of convention. Why the transpose?

A matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ is map transforming vectors in an n-dimensional space $\mathbb{R}^n$ to a vector in an m-dimensional space $\mathbb{R}^m$, and by convention we typically write $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$ transforming column vectors into other column vectors. We can think of the n-dimensional vector $\boldsymbol{x}$ being shoved into the matrix from the top (n, the number of columns, is the number of inputs). $\boldsymbol{x}$ churns through the matrix $\boldsymbol{A}$ and is ultimately thrown out as a new m-dimensional vector from the

left of the matrix:

$$\begin{array}{rc} & \text{Inputs} \quad \leftarrow \\ \text{Outputs} \leftarrow & \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \end{array} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

Conveniently, we can think of the Jacobian matrix $\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{x}}$ as having the same interpretation. The inputs line the top of the matrix, and the outputs line the left side of the matrix. The ijth entry of the matrix itself tells us how tweaking the jth input affects the i output of the original function; again we have inputs corresponding to the j index (columns) and outputs corresponding to the i index (rows).

This input-output convention fits very nicely with the way the chain rule works. Suppose now there's another function $\boldsymbol{h} : \mathbb{R}^k \to \mathbb{R}^n$ that takes k-dimensional vectors $\boldsymbol{u} \in \mathbb{R}^k$ to vectors $\boldsymbol{x} \in \mathbb{R}^n$ in the domain (input space) of $\boldsymbol{g}$. For each g_i the chain rule says

$$\frac{\partial}{\partial u_k} g_i(h(\boldsymbol{u})) = \sum_{j=1}^{n} \frac{\partial g_i}{\partial x_j} \frac{\partial h_j}{\partial u_k}, \tag{18}$$

so if you squint hard enough you can convince yourself that the individual Jacobians of $\boldsymbol{g}$ and $\boldsymbol{h}$ relate to the overall Jacobian of the composed function $\boldsymbol{f} = \boldsymbol{g} \circ \boldsymbol{h} : \mathbb{R}^k \to \mathbb{R}^m$ in the following way[1]

$$\frac{\partial}{\partial \boldsymbol{u}} (\boldsymbol{g} \circ \boldsymbol{h}) = \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{x}} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{u}}. \tag{19}$$

Note that the matrices line up nicely without having to take any transposes, and we could have continued the chain further to the right to account for as many function compositions as necessary since the rule works recursively.

Now, that's convenient. The matrix $\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{u}}$ tells us how h_j changes when we tweak u_k, and then in turn $\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{x}}$ tells us how the final output g_i changes when the input x_j, which h_j feeds into, changes. Since the outputs of $\boldsymbol{h}$ become the inputs of $\boldsymbol{g}$, the derivatives align such that the outputs of $\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{u}}$ become the inputs of $\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{x}}$. This relation is furthered when take time derivatives (which, really, we can also view as just another application of the chain rule since $\boldsymbol{u}$ is now a vector valued function of time):

$$\frac{d}{dt} (\boldsymbol{g} \circ \boldsymbol{h})(\boldsymbol{u}) = \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{x}} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{u}} \dot{\boldsymbol{u}}. \tag{20}$$

We can again think through this formula intuitively in terms of the inputs and outputs of the constituent matrices. $\boldsymbol{u}(t)$ is the first function we encounter

[1] See Appendix A for a high level discussion of why the chain rule becomes matrix multiplication.

in the composition, so it makes sense that the rightmost factor in the time derivative tells us how the underlying inputs to the entire system are changing with time. Then, since $\boldsymbol{u}$ feeds into $\boldsymbol{n}$ and in turn the inputs to $\boldsymbol{h}$ churn through the function to become the outputs of $\boldsymbol{h}$, the middle term tells us how changes to the inputs of $\boldsymbol{h}$ become changes to the outputs of $\boldsymbol{h}$. Combined, $\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{u}}\dot{\boldsymbol{u}}$ tells us how the changes in $\boldsymbol{u}$ with respect to time, which are changes to the inputs of $\boldsymbol{h}$, become changes to the outputs of $\boldsymbol{h}$. And finally, since outputs of $\boldsymbol{h}$ feed into $\boldsymbol{g}$ as inputs, the final multiplication by $\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{x}}$ tells us how those changes to the outputs of $\boldsymbol{h}$ ultimately become changes to the outputs of $\boldsymbol{g}$. The outputs of $\boldsymbol{g}$ are the outputs of the entire system, so we're done.

In other words, since matrices take inputs along the top to outputs along the left side, the way in which inputs lead to outputs in $\boldsymbol{g} \circ \boldsymbol{h}(\boldsymbol{u})$ dictates the final structure of how the matrices of partial derivatives should be laid out. The convention we chose above allows us to leverage our intuition of how a matrix should behave to understand the flow of information through the chain rule.

It's important to remember, though, that when $f : \mathbb{R}^n \to \mathbb{R}$, the Jacobian of this one-output map is actually the transpose of the gradient. So when we have a second function $\boldsymbol{g} : \mathbb{R}^m \to \mathbb{R}^n$ and we nest the two we can write all of the following

$$\begin{aligned}\frac{\partial}{\partial \boldsymbol{x}} f(\boldsymbol{g}(\boldsymbol{x})) &= \frac{\partial f}{\partial \boldsymbol{u}}(\boldsymbol{g}(\boldsymbol{x}))\ \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{x}}(\boldsymbol{x}) = \nabla_{\boldsymbol{u}} f(\boldsymbol{g}(\boldsymbol{x}))^T\ \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{x}}(\boldsymbol{x}) \\ &= \left(\left[\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{x}}(\boldsymbol{x})\right]^T \nabla_{\boldsymbol{u}} f(\boldsymbol{g}(\boldsymbol{x})) \right)^T = \Big(\nabla_{\boldsymbol{x}} f(\boldsymbol{g}(\boldsymbol{x})) \Big)^T,\end{aligned}$$

where we're denoting the input of f by $\boldsymbol{u} = \boldsymbol{g}(\boldsymbol{x})$.

4.1 First-order Taylor for multi-dimensional maps

Here we just note that when taking the Taylor expansions of multi-dimensional multi-output maps, we can simply reduce the map to its individual component functions. A map $\boldsymbol{g} : \mathbb{R}^n \to \mathbb{R}^m$ can be expressed in terms of its m output components as $\boldsymbol{g}(\boldsymbol{x}) = [g_1(\boldsymbol{x}), \ldots, g_m(\boldsymbol{x})]^T$. Each of these functions, individually, have a Taylor expansion, so we can use these component expansions in combination as a Taylor expansion for the full map. For instance, the first order Taylor expansion around $\boldsymbol{u} \in \mathbb{R}^n$ is

$$\boldsymbol{g}(\boldsymbol{x}) \approx \begin{bmatrix} g_1(\boldsymbol{u}) + \nabla g_1(\boldsymbol{u})^T(\boldsymbol{x} - \boldsymbol{u}) \\ g_2(\boldsymbol{u}) + \nabla g_2(\boldsymbol{u})^T(\boldsymbol{x} - \boldsymbol{u}) \\ \vdots \\ g_m(\boldsymbol{u}) + \nabla g_m(\boldsymbol{u})^T(\boldsymbol{x} - \boldsymbol{u}) \end{bmatrix} \tag{21}$$

$$= \boldsymbol{g}(\boldsymbol{u}) + \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{x}}(\boldsymbol{u})\ \Big(\boldsymbol{x} - \boldsymbol{u} \Big). \tag{22}$$

5 Calculations: Tricks of the trade

There are a number of resources in books or online that list vector and matrix calculus formulas. Those formulas are useful to know and they can save time, but perhaps more importantly, it's often critical to be fluent in the technique of boiling the problem down to its simplest terms, its components, taking the partials, and then reconstructing the result in a convenient matrix form. This method of deconstruction-reconstruction is what is often use to derive the basic formulas, as we'll see in Section 5.1, and in cases where none of your formulas are applicable, you can always still use this method to boil it down to the basics. This technique is especially important when chain rules result in higher-order tensors of derivatives that we haven't discussed and that you may not be familiar with. Despite their unfamiliarity, derivatives are derivatives, and you can always use this technique to systematically deconstruct the structure of a calculation and come up with an efficiently implementable formula for its derivatives.

5.1 Deriving formulas

Two types of terms that arise in a number of situations are linear terms of the form $\boldsymbol{b}^T\boldsymbol{x}$ and quadratic terms of the form $\frac{1}{2}\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x}$. We'll use these simple terms to step through what we call the deconstruction-reconstruction technique.

The basic idea behind deconstructing the derivative is to calculate the individual components of the gradient, Hessian, or Jacobian and then stare at the resulting expression and see if it can be conveniently packed back into vector-matrix notation. Consider, for instance, the simple linear term $\boldsymbol{b}^T\boldsymbol{x}$. When finding the gradient of this linear term, we can look at the individual partials with respect to each variable separately. Since the only term that has anything to do with the kth input is the kth term,

$$\frac{\partial}{\partial x_k}\sum_{i=1}^{n} b_i x_i = \frac{\partial}{\partial x_k} b_k x_k = b_k. \tag{23}$$

We finish by stacking the partials into vector form to see what the gradient is in full:

$$\nabla \boldsymbol{b}^T\boldsymbol{x} = \begin{bmatrix} \frac{\partial}{\partial x_1}\sum_{i=1}^{n} b_i x_i \\ \frac{\partial}{\partial x_2}\sum_{i=1}^{n} b_i x_i \\ \vdots \\ \frac{\partial}{\partial x_n}\sum_{i=1}^{n} b_i x_i \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} = \boldsymbol{b}. \tag{24}$$

Thus, we can summarize our work into the formula $\nabla \boldsymbol{b}^T\boldsymbol{x} = \boldsymbol{b}$.

That was a simple example, but a slightly more complicated example is the quadratic term $\frac{1}{2}\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x}$. We'll assume for this one that $\boldsymbol{A}$ is symmetric and compute both the gradient and the Hessian. For the gradient, we need to first

partials derivatives. Denoting the components of $\boldsymbol{A}$ as a_{ij}, we get

$$\begin{aligned}\frac{\partial}{\partial x_k}\frac{1}{2}\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} &= \frac{1}{2}\frac{\partial}{\partial x_k}\sum_{i=1}^{n}\sum_{j=1}^{n}a_{ij}x_ix_j \\ &= \frac{\partial}{\partial x_k}a_{kk}x_k^2 + \frac{1}{2}\frac{\partial}{\partial x_k}\sum_{i\neq k}a_{ik}x_ix_k + \frac{1}{2}\frac{\partial}{\partial x_k}\sum_{k\neq j}a_{kj}x_kx_j \\ &= a_{kk}x_k + \sum_{i\neq k}a_{ik}x_i = \sum_{j=1}^{n}a_{kj}x_j.\end{aligned}$$

And stacking these components into a vector, we get

$$\nabla\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} = \begin{bmatrix}\sum_{j=1}^{n}a_{1j}x_j \\ \sum_{j=1}^{n}a_{2j}x_j \\ \vdots \\ \sum_{j=1}^{n}a_{nj}x_j\end{bmatrix} = \boldsymbol{A}\boldsymbol{x}. \tag{25}$$

Again, we can summarize the result as simply as a formula $\nabla\left(\frac{1}{2}\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x}\right) = \boldsymbol{A}\boldsymbol{x}$. If we remember this formula and the fact that it's only applicable when $\boldsymbol{A}$ is symmetric and there's a factor of $\frac{1}{2}$ out in front, then great, we can use it. But if we've forgotten any of that, we can always re-derive it by re-calculating the partial derivatives and stacking them.

Finally, we can easily calculate the formula for the Hessian of $\frac{1}{2}\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x}$ by examining the second partial:

$$\frac{\partial^2}{\partial x_l\partial x_k}\left(\frac{1}{2}\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x}\right) = \frac{\partial}{\partial x_l}\left(\sum_{j=1}^{n}a_{kj}x_j\right) = \frac{\partial}{\partial x_l}\left(\,a_{kl}x_l\,\right) = a_{kl}. \tag{26}$$

Collecting all of these second partials into a matrix tells us that $\nabla^2\left(\frac{1}{2}\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x}\right) = \boldsymbol{A}$.

5.2 A more complicated example: Derivatives of a norm

Here we'll take a look at a somewhat more complicated example: the derivatives of a norm $f(\boldsymbol{x}) = \|\boldsymbol{x}\|$. It's an interesting exercise to try to apply formulas to calculating these derivatives in vector and matrix form directly. If you're skilled at applying the rules, then it's fairly straightforward. But if not, or if you're rusty, it's very easy to mess up transposes and such and to make little convention mistakes that entirely destroy your result. This section returns to the basics and demonstrates that simply deriving the partial derivatives and recognizing the result is an effective way to find both the gradient and the Hessian, and to be sure you're right.

We start by rewriting $f(\boldsymbol{x}) = \|\boldsymbol{x}\| = \left(\sum_{i=1}^{n} x_i^2\right)^{\frac{1}{2}}$. Now we can easily calculate the first partial derivatives

$$\begin{aligned}\frac{\partial}{\partial x_k} f(\boldsymbol{x}) &= \frac{\partial}{\partial x_k}\left(\sum_{i=1}^{n} x_i^2\right)^{\frac{1}{2}} \\ &= \frac{1}{2}\left(\sum_{i=1}^{n} x_i^2\right)^{-\frac{1}{2}} 2x_k \\ &= \frac{x_k}{\|\boldsymbol{x}\|}.\end{aligned}$$

Stacking these in vector form gives $\nabla\|\boldsymbol{x}\| = \frac{\boldsymbol{x}}{\|\boldsymbol{x}\|} = \widehat{\boldsymbol{x}}$.

Now continuing on to the Hessian, the second partial of this function is

$$\begin{aligned}\frac{\partial}{\partial x_l}\frac{x_k}{\|\boldsymbol{x}\|} &= \frac{\|\boldsymbol{x}\|\frac{\partial x_k}{\partial x_l} - x_k\frac{\partial}{\partial x_l}\|\boldsymbol{x}\|}{\|\boldsymbol{x}\|^2} \\ &= \frac{1}{\|\boldsymbol{x}\|}\left(\frac{\partial x_k}{\partial x_l} - \frac{x_k}{\|\boldsymbol{x}\|}\frac{x_l}{\|\boldsymbol{x}\|}\right).\end{aligned}$$

We can now recognize that the partial of x_k with respect to x_l is 1 when the variables are the same (i.e. when $k = l$) and 0 when they aren't ($k \neq l$), we see that the matrix with elements $\frac{x_k}{x_l}$ is just $\boldsymbol{I}$. Moreover, in general, the matrix formed of elements $u_i\, u_j$ is given by the outer product $\boldsymbol{u}\,\boldsymbol{u}^T$, where the vector $\boldsymbol{u}$ consists of elements u_i. So the second term inside the parentheses is just $\widehat{\boldsymbol{x}}\,\widehat{\boldsymbol{x}}^T$, where $\widehat{\boldsymbol{x}} = \frac{\boldsymbol{x}}{\|\boldsymbol{x}\|}$. Combining these observations, we see that in matrix form, the Hessian is

$$\nabla^2\|\boldsymbol{x}\| = \frac{1}{\|\boldsymbol{x}\|}\left(\,\boldsymbol{I} - \widehat{\boldsymbol{x}}\,\widehat{\boldsymbol{x}}^T\,\right) \tag{27}$$

The nice thing about vector and matrix notation is that it's easier to see the geometry behind the result. In this case, the gradient is just the normalized vector $\widehat{\boldsymbol{x}}$, so it's everywhere just the unit vector pointing away from the origin (see Figure 4).

The Hessian here has a nice geometric interpretation, too. Generally, the Hessian defines how the function is curved in various directions. In this case, notice that the vector $\widehat{\boldsymbol{x}}$ is an Eigenvector with Eigenvalue 0 since $(\boldsymbol{I} - \widehat{\boldsymbol{x}}\,\widehat{\boldsymbol{x}}^T)\widehat{\boldsymbol{x}} = \widehat{\boldsymbol{x}} - \widehat{\boldsymbol{x}} = \boldsymbol{0}$. In other words, along that single dimension, the function isn't curving at all, it's flat (linear). That makes intuitive sense since the graph of function forms a cone (see Figure 4) and this dimension slices the cone straight to its tip at the origin. However, any vector $\boldsymbol{v}$ orthogonal to $\widehat{\boldsymbol{x}}$ is also an Eigenvector but with Eigenvalue 1, since

$$\left(\,\boldsymbol{I} - \widehat{\boldsymbol{x}}\,\widehat{\boldsymbol{x}}^T\,\right)\boldsymbol{v} = \boldsymbol{v} - \widehat{\boldsymbol{x}}\,\underbrace{(\widehat{\boldsymbol{x}}^T\boldsymbol{v})}_{=0} = \boldsymbol{v}.$$

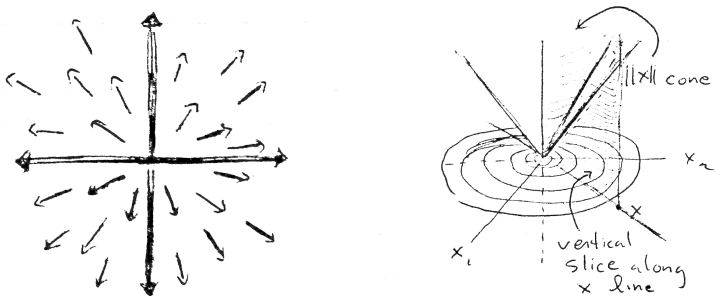

Figure 4: **Left:** The gradient of the norm function is everywhere a unit vector pointing away from the origin. **Right:** Its Hessian in the direction pointing toward the origin is zero due to the conic shape of the function. And the Hessian orthogonal to that direction represents the isometric curvature inherent in the isocontours.

Thus, along those directions, the function is curving isometrically, as though that quadratic term were simply $\frac{1}{2}\boldsymbol{x}^T\boldsymbol{I}\boldsymbol{x}$. Examining the figure, we see that these dimensions are those hit the strongest by the spherical curvature of the isocontours of the function.

5.3 Robot manipulator Jacobian

Calculating derivatives is a great skill to have, and it's one that'll take some time to become fluent with. But blind calculation can be unduly tedious, and at times it can result in cumbersome, opaque, expressions that are both difficult to implement and difficult to debug. This section gives an example of where a strong understanding of what derivatives are can lead to geometric insight into problems that greatly simplify the calculation of otherwise very difficult derivatives. Specifically, we examine the Jacobian of a revolute robot manipulator, such as that shown in Figure 5, and develop the geometric intuition behind its calculation.

Suppose each pair of links (body parts) of the manipulator are connected by a revolute (rotational) joint. Each link is rigid (doesn't bend) and each joint rotates around an axis fixed relative to the link (see the top right plot of Figure 5). We often model such a manipulator as a nonlinear map $\phi : \mathbb{R}^n \to \mathbb{R}^3$, mapping the n joint angles to the three-dimensional location of the end-effector. In the discussion below we'll denote the vector of joint angles as $\boldsymbol{q} \in \mathbb{R}^n$.

The Jacobian tells us how the outputs of a map change with little changes to the inputs. In this case, the inputs are the joint angles, and the outputs are the three-dimensional end-effector position. More specifically, the kth joint angle is the angle of rotation around the kth joint axis, so the kth column of the Jacobian tells us how the position of the end-effector changes when all joints are kept fixed except the kth joint, which moves a tiny bit around the physical axis.

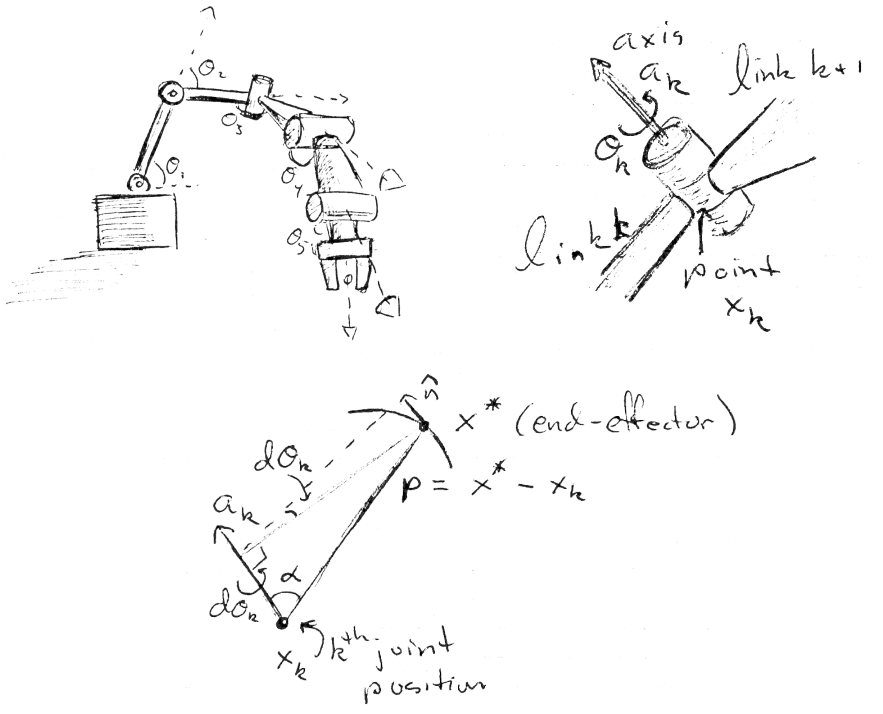

Figure 5: **Top left:** A three-dimensional fixed-based revolute manipulator. **Top right:** A closeup on two links connected by a revolute axis. The axis $\boldsymbol{a}_k$ is rooted at point $\boldsymbol{x}_k$, and the vector pointing from $\boldsymbol{x}_k$ to the end-effector $\boldsymbol{x}^*$ is $\boldsymbol{p} = \boldsymbol{x}^* - \boldsymbol{x}_k$. **Bottom:** The geometry of rigid rotation around a fixed axis. The details of the robot links between the kth joint and the end-effector are stripped away because they're irrelevant to the geometry.

Since all other joints remain fixed, both the section of the robot between the base (connection to the ground) and the kth joint axis, and the section of the robot between the kth joint axis and the end-effector, are entirely rigid. Thus, the small movement $d\theta_k$ around the kth axis doesn't change the position of the axis at all, just rotates the rigid portion of the robot downstream from the joint (i.e. between the joint and the end-effector) by some small amount.

Let $\boldsymbol{a}_k$ be a unit vector denoting the physical configuration of the kth joint axis emanating from the physical location $\boldsymbol{x}_k$ of the k joint as seen by (i.e. in the coordinate system of) the world. The geometry of the setup dictates that the resulting motion of the end-effector is entirely governed by the orthogonal rotation of the point around the physical axis in question (see Figure 5). Intuitively, if we were to rotate the entire world around the line formed by the axis in question, all points will rotate in circles around the line through the

plane orthogonal to the axis containing the point. That's simply the geometry of such a rotation around the axis. The exact same thing happens with the robot, and, in particular, with the specific end-effector point on the robot. It rotates through a circle around the axis within a plane orthogonal to the line of the axis as depicted in the figure.

Suppose the angle increases counterclockwise around the axis, and let $\widehat{\boldsymbol{n}}$ be a unit vector tangent to the circle of rotation pointed in the direction of positive increase. This unit vector is also orthogonal to the plane formed by the axis and the vector pointing from the joint to the end-effector $\boldsymbol{p} = \phi(\boldsymbol{q}) - \boldsymbol{x}_k$. That point will be of significance later. Denote the angle between $\boldsymbol{a}_k$ and $\boldsymbol{p}$ by α.

The small motion $d\theta_k$ is amplified by the radial distance of the end-effector point away from the axis. In this case, the point, which is a radial distance $r = \|\boldsymbol{p}\| \sin\alpha$ away from the axis $\boldsymbol{a}_k$, will move a magnitude $d\theta_k \|\boldsymbol{p}\| \sin\alpha$ in the direction $\widehat{\boldsymbol{n}}$ when rotated by $d\theta_k$. Thus, the ratio of the end-effector movement to the amount of movement around the axis $d\theta_k$ is

$$\frac{\partial \phi}{\partial \theta_k} = \frac{1}{d\theta_k} \Big(d\theta_k \|\boldsymbol{p}\| \sin\alpha \Big) \widehat{\boldsymbol{n}} = \boldsymbol{a}_k \times \boldsymbol{p}, \tag{28}$$

where that final expression shows that this geometric calculation miraculously somehow matches the definition of cross product which we know to be $\boldsymbol{a} \times \boldsymbol{b} = \|\boldsymbol{a}\|\|\boldsymbol{b}\| \sin\alpha$ when α is the angle between $\boldsymbol{a}$ and $\boldsymbol{b}$. Actually, it's not so miraculous. Coincidences like this are never really coincidences in mathematics, so clearly there's something deeper going on, but the derivation we give here is a good way to find the result without having to understand the deeper significance of its connection to the cross product.

The bottom line is that now we have an algorithm for calculating the Jacobian that has nothing to do with the typical mechanics of differentiation. Given a configuration of joint angles $\boldsymbol{q} \in \mathbb{R}^n$ we simply need to compute the three-dimensional joint locations $\boldsymbol{x}_k$ and the corresponding (normalized) axis directions $\boldsymbol{a}_k$ in world coordinates, along with the location of the end-effector $\boldsymbol{x}^* = \phi(\boldsymbol{q})$, and then the cross products $\boldsymbol{a}_k \times (\boldsymbol{x}^* - \boldsymbol{x}_k)$ give us the columns of the Jacobian. The columns, by definition, tell us how the end-effector changes with small changes to a given joint, and as we saw above through our geometric argument this cross product precisely gives that rate of change.

The actual mechanics of calculating the necessary rotation matrices and translation vectors (affine transformations) necessary to find the three-dimensional world positions and orientations of the axes is beyond the scope of this document, but there are a number of texts that discuss that portion of the computation in detail (see, for instance, Siciliano et al. (2010)). The resulting computation is relatively straightforward and very efficient; this algorithm is used in most practical state-of-the-art implementations of forward kinematics.

5.4 Checking your work with finite-differences

As a final, but very important note, remember that we can always check that we've implemented all of these partial derivatives correctly by checking them

numerically on a computer. The partial derivative is explicitly a measure of how fast the function is changing when one of the inputs is tweaked by an ϵ amount. The derivative itself is a limit as the ϵ becomes smaller and smaller, but if we just want to check our work, we can always approximate the derivative numerically by choosing a small finite ϵ and computing

$$\frac{d}{dx} f(x) \approx \frac{f(x+\epsilon) - f(x)}{\epsilon} \quad \text{or} \quad \frac{f(x+\frac{\epsilon}{2}) - f(x-\frac{\epsilon}{2})}{\epsilon}. \tag{29}$$

For small epsilon we get a pretty good estimate of what the partial derivative should be directly from the implementation of the function value, so we can be sure that if our direct derivative implementation matches the numerical approximation, it's also consistent with the function value calculation. We can compute approximations of first and second partial derivatives, too, simply by applying the same trick to the respective components.

Why don't we just use these finite-differences in practice? When we're just differentiating with respect to a single variable, such as time, then the overhead isn't bad. And we may even be able to reuse some of the computation if, for instance, we're finding time derivatives sequentially for multiple times between time end points 0 and T. In those cases, finite-differencing may be a practical and computationally efficient alternative to calculating and manipulating analytical derivatives. In general, though, for n-dimensional functions, we need $O(n)$ function evaluations to compute the approximation, which is rather hefty overhead given that the analytical gradient can usually be calculated in just a constant multiple of the time it takes to compute a single function evaluation. Moreover, second derivatives are even worse in terms of there computational overhead, and since we're taking finite-differences of finite-differences in that case, they can easily become numerically unstable. Nevertheless, they're a very effective way to verify that we haven't made any mistakes somewhere in the often tedious pipeline between deriving and implementing the derivatives. And that makes them an invaluable tool for system engineering.

References

Siciliano, Bruno, Sciavicco, Lorenzo, Villani, Luigi, and Oriolo, Giuseppe. *Robotics: Modelling, Planning and Control*. Springer, second edition, 2010.

Appendix

A Why the chain rule really becomes matrix multiplication

Most Advanced Calculus textbooks present the appearance of matrix multiplication in the chain rule almost as though it were a magical, coincidental,

fortuitous, inexplicably serendipitous result that just happened to work out because the stars aligned and fate smiled down at that first brave soul bold enough to tackle writing it out. But, as with most apparent coincidences in math, that's not the case. It's a manifestation of something more fundamental.

Advanced analysis text, especially those leading up to calculus on manifolds, present a lot of the rules of calculus using a basis-free approach, such as the abstract development we've seen for linear algebra. That means rather than representing n-dimensional vectors using collections of n numbers, the expositions leave them simply as abstract elements of a set that satisfies enough axioms to be considered a linear vector space. If we so choose, we can always represent a vector using the n coefficients of that vector's description in some n-dimensional basis for the space, but since the choice of basis is largely arbitrary, these texts study the underlying properties of linear algebra and calculus abstractly as much as possible without fixing a particular basis in advance.

In this setting, the abstract chain rule says that the total derivative of a composed map $\boldsymbol{g} \circ \boldsymbol{h}$ is a linear transform consisting of a product of the linear transforms that come individually from $\boldsymbol{g}$ and $\boldsymbol{h}$. Fundamentally, the chain rule just relates overall derivatives to products of constituent derivatives. Once we ultimately calculate these results for particular choices of bases, products of linear transforms then manifest as matrix multiplication rules that tell us how the *coefficients* of a vector in the domain turn into *coefficients* of the transformed vector in the range. Thus, our matrix multiplication of Jacobian matrices is actually just a bookkeeping trick to track how coefficients of particular vector representations transform under the more fundamental basis-free linear transformations representing the underlying derivatives.

Matrix multiplication doesn't just happen to work out because it was a clever way to compactly represent all the sums that crop up when applying the chain rule to composed vector-valued multivariate functions, the matrix multiplication arises because it represents how the coefficients under our particular (Cartesian) choice of bases transform under the abstract chain rule governing the process.