Multivariate Calculus II: The geometry of smooth maps

Nathan Ratliff

Nov 28, 2014

Abstract

This document explores the geometry of smooth maps, building many of the ideas from an analysis of attractor algorithms across surface representations. Through geometric intuition and algebraic analysis of these algorithms, we see how geometry may be pulled back from the co-domain of a smooth nonlinear map to construct a consistent representation of that geometry in the map's domain, and we study how we might construct algorithms that build on geometric or physical properties of a system so that their behavior is agnostic to the particular choice of representation. These tools and ideas originate in differential geometry and a study of Riemannian manifolds; this document emphasizes intuition over rigorous construction, particularly as demonstrated by the behavior of first-order Taylor expansions. In all cases, the map's Jacobian remains key to understanding how geometry transforms from one space to another.

1 Nested mappings in practice

It's tempting to see linear spaces everywhere since linear algebra is such a powerful tool. Research areas of machine learning, optimization, and robotics are littered with applications of linear algebra. When we find a linear structure in a problem we're very happy since suddenly we know a lot about its geometry. But many problems, especially in robotics, aren't really linear by nature, so at some point we need to acquiesce to reality and address smooth maps. Robots themselves, for instance, as we'll see in detail later on in Section 4, can be understood as smooth maps from their space of joint values to the real 3D world. And those maps can be highly nonlinear in practice.

Similarly, in machine learning most problems revolve around finding the right function to represent some data. And to simplify things we often assume that all of those functions $y = f(\boldsymbol{x})$ are linear in some unknown parameter vector $\boldsymbol{w}$ (i.e. $y = \boldsymbol{w}^T\boldsymbol{x}$ for some $\boldsymbol{w}$)—we task the algorithm, then, with finding the right $\boldsymbol{w}$ to best represents the data. That makes for some nice math, but it also severely restricts the tools we produce. That assumption presumes we, the designers or engineers, know what parts or features of the problem are important for representing the data. The algorithm really only ends up automating

only the process of fiddling with coefficients on the function to weigh the parts together properly. That's exceedingly useful when there are many parts, but in reality, especially when we're building machine learning methods to understand unstructured inputs like natural images or videos, we really have little idea how to create such a collection of features—the problem is just too hard. So increasingly researchers are interested in understanding how we can learn a sequence of smooth transformations of the inputs (e.g. the image) to ultimately convert the input into a representation more amenable to all of our linear algorithms. Suddenly we're forced to understand better the geometry[1] of chained nonlinear maps.

Chains of nonlinear maps like those mentioned above are found all over the place when working with intelligent systems, so it's important that we have a strong understanding of how geometry changes from one space to the next when we apply nonlinear maps so we can get a handle on how they affect the behavior of the system and the algorithms designed for it. The inverse problem is particularly important—how might we choose the geometry of a domain space so that *when* we transform it by a nonlinear map the resulting geometry is the geometry that we really care about.

This document addresses the above questions in detail, stepping first through intuitive geometric arguments constructed around surface representations, before introducing a powerful generalization that leverages ideas from Riemannian geometry. The first-order geometry of nonlinear maps, as embodied by their Jacobians, plays a central role in all of these ideas; we see repeatedly that the Jacobian and, specifically, the first-order Taylor expansion of a nonlinear map connects us back to our familiar tools from linear algebra and gives us something we can understand, analyze, and tangibly manipulate.

2 The geometry of implicit surfaces

Lets start by analyzing the geometry of a particular type of surface that arises quite frequently in nonlinear optimization: the implicit surface. An implicit surface is one defined by a smooth mapping $g : \mathbb{R}^n \to \mathbb{R}^k$ as the set of all points satisfying $g(\boldsymbol{x}) = \mathbf{0}$. The top row of Figure 1 shows two examples of such surfaces, one generated as the zero set of a single output function in three-dimensions, and the other generated as the zero set of a two output function in the same space. Notice that, in general, the dimension of the surrounding space, which we typically term the **ambient space**, minus the number of outputs seems to define the dimensionality of the surface. In the first case, we have one output in three dimensions, so the surface is two dimensional. And in the second case, we have two outputs in three dimensions and the surface is one dimensional. Why is that?

[1] Throughout this document, when we say "geometry" we mean the formal notion of inner product (angle) and, especially, norm in the space.

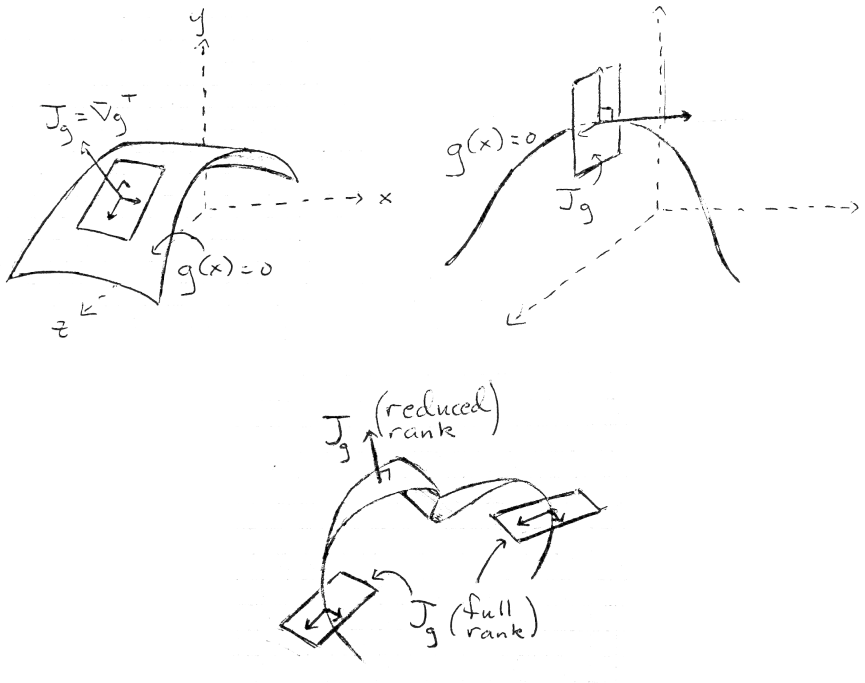

Figure 1: **Top left:** A 2D implicit surface in 3D defined by $g : \mathbb{R}^3 \to \mathbb{R}$. **Top right:** A 1D implicit surface in 3D defined by $g : \mathbb{R}^3 \to \mathbb{R}^2$. **Bottom:** A mixed dimensionality implicit surface in 3D defined by $g : \mathbb{R}^3 \to \mathbb{R}^2$ with a region in the middle where the Jacobian reduces rank.

2.1 The Jacobian tells us everything

We can gain insight into the local geometry of the surface at a given point $\boldsymbol{x}_0$ by considering the surface's first-order Taylor expansion. Consider the generic case, where we have k outputs in an n-dimensional domain. The first-order Taylor expansion is

$$g(\boldsymbol{x}) \approx \underbrace{g(\boldsymbol{x}_0)}_{=\mathbf{0}} + \underbrace{\frac{\partial g}{\partial \boldsymbol{x}}(\boldsymbol{x}_0)}_{\boldsymbol{J}_g} \underbrace{\Big(\boldsymbol{x} - \boldsymbol{x}_0 \Big)}_{\delta \boldsymbol{x}} = \boldsymbol{J}_g \delta \boldsymbol{x}.$$

This expansion defines to the first order how the map varies as we move a little ways $\delta\boldsymbol{x}$ away from the expansion point $\boldsymbol{x}_0$. The behavior is entirely defined by the Jacobian $\boldsymbol{J}_g = \frac{\partial g}{\partial \boldsymbol{x}}(\boldsymbol{x}_0)$. In this case, the surface is the set of all points for which the value of the map is the constant value $\mathbf{0}$. As indicated in the equation, the value $g(\boldsymbol{x}_0)$ is already $\mathbf{0}$, so to understand the local shape and dimensionality of the surface, we want to examine the space of directions $\delta\boldsymbol{x}$ we could move in that don't change the value of the map. Specifically, we want to

find the space of the form

$$\mathcal{T}_{\boldsymbol{x}_0} = \left\{ \delta\boldsymbol{x} \mid \boldsymbol{J}_g \delta\boldsymbol{x} = \mathbf{0} \right\}.$$

But we already know what this space is! It's precisely the (right) null space of $\boldsymbol{J}_g$. The space of all *perturbations* that don't change the value of the map (to the first order) as we move away from $\boldsymbol{x}_0$ is a space known as the **tangent space** to the surface at $\boldsymbol{x}_0$. For implicit surfaces, as we just saw, the tangent space *is* the null space of the map's Jacobian.

We know that the null space, as long as the Jacobian is full rank, has dimension $n - k$. And that explains why the first two pictured surfaces are of dimension 2 and 1, respectively. Note that, in general, there may be points where $\boldsymbol{J}_g$ isn't full rank. In that case, the dimensionality of the null space is larger. If our map has k outputs, but only rank $l < k$, the dimensionality of the null space is $n - l > n - k$. Thus, we can find surfaces such as that pictured in the lower subplot of Figure1. In that case, the implicit surface is represented by a map from $\mathbb{R}^3$ to the one-dimensional space $\mathbb{R}$, but the Jacobian is full rank (rank 2) only in a small region of the space where we see the surface manifesting as a one-dimensional trajectory. Everywhere else the Jacobian has rank 1 and the surface expands to two-dimensions.

Likewise, the space orthogonal to that tangent space must be the Jacobian's row space. For the pictured 2D surface in 3D (top left), that rows space is just one-dimensional and is given by the gradient of the function (which is the transposed Jacobian in this case). And for the pictured 1D surface in 3D (top right), that orthogonal space, the row space, is two-dimensional as depicted in the image.

Again, we see that Jacobian of a nonlinear map tells us a lot about the local geometry of the mathematical object in question, in this case, the surface. In particular, an SVD of the Jacobian explicitly gives orthogonal representations of these spaces of interest:

$$\boldsymbol{J}_g = \begin{bmatrix} \boldsymbol{U}_{/\!/} & \boldsymbol{U}_{\perp} \end{bmatrix} \begin{bmatrix} \boldsymbol{S} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{V}_{/\!/}^T \\ \boldsymbol{V}_{\perp}^T \end{bmatrix}.$$

$\boldsymbol{V}_{\perp}$ spans the null space (the tangent space $\mathcal{T}_{\boldsymbol{x}_0}$) and $\boldsymbol{V}_{/\!/}$ spans the row space ($\mathcal{T}_{\boldsymbol{x}_0}$'s orthogonal compliment). If we're just looking for a robust way to manipulate these quantities, then we can always just take the SVD of the matrix and play with those two blocks of right singular vectors. We do exactly that in the next section to demonstrate how we can exploit the local geometry of the surface as expressed through the Jacobian. But Section 3 gives some detail on how we can play around with these quantities a little more efficiently in practice when computational complexity might really matter.

2.2 Following potential fields across an implicit surface

In this section, we use straightforward geometric arguments to derive a simple algorithm that walks across an implicitly defined surface following as well as

possible the vector field defined by a potential function. This notion of *potential function* comes from physics where we see the idea manifest as *potential energy*, but various areas of intelligent systems also borrow the concept so it's good to be familiar with the terminology. Smooth potential functions define vector fields called *potential fields* (in physics these vectors are forces) pointing along paths that converge to a local minimum[2] of the potential function. Specifically, the vector directions are given by the negative gradients of the potential function. For instance, the gravitational potential is a potential that increases linearly with height (assuming gravity is constant), so the corresponding force (gravity) is a uniform vector field always pointing downward with the same magnitude ($9.8m/s^2$ in SI units).

In robotics, we see potential functions entering into models of control as attractors that pull the end-effector in various directions, such as to a designated point. Such a potential functions live in the end-effector's workspace and the potential vectors must be translated to the space of joint angles to define how to properly command the robot to move in the right direction. Section 4.2 studies that problem in detail, but specific applications aside, this section addresses the generic problem of devising an algorithm to walk across an implicit surface through the ambient space following a potential field.

Let $\psi : \mathbb{R}^n \to \mathbb{R}$ be a potential function defining a vector field of negative gradient vectors $\boldsymbol{v}(\boldsymbol{x}) = -\nabla\psi(\boldsymbol{x})$. We want to somehow follow this vector field to the best of our ability while sticking to the implicit surface.

From our above analysis, the geometry of this problem is clear. The tangent space at any given point $\boldsymbol{x}_t$ defines, to the first-order, the space we want to stay in. We can't willy-nilly take steps in the direction $\boldsymbol{v}(\boldsymbol{x}_t)$, but we *can* project that vector onto this tangent space and take a little step in the resulting direction. As long as our step is small, the tangent space is a good approximation to the surface at $\boldsymbol{x}_t$, so the resulting algorithm will stick to the surface pretty well[3] while following the vector field to the best of its ability.

What specifically then is the step? If we're at $\boldsymbol{x}_t$, we can calculate the Jacobian $\boldsymbol{J}_g$ at that point. An SVD of $\boldsymbol{J}_g$ gives us a collection of right singular vectors packed into a matrix $\boldsymbol{V}_\perp$ that span the null space; projecting onto those vectors is simply a matter of multiplying by the matrix $\boldsymbol{V}_\perp\boldsymbol{V}_\perp^T$ (the transposed matrix finds the expansion coefficients, and then the other matrix applies them to the singular vectors). Thus, the final step direction becomes $\widetilde{\boldsymbol{v}}(\boldsymbol{x}_t) = \left(\boldsymbol{V}_\perp\boldsymbol{V}_\perp^T\right)\boldsymbol{v}(\boldsymbol{x}_t)$

[2]Intuitively, a local minimum is the bottom-most point of any bowl-like region of the function. These ideas are made explicit in the study of optimization.

[3]The algorithm we describe here is a form of numerical differential equation solver called Euler integration. As you can imagine, it has a tendency to drift away from strongly curved surfaces since it's always just following the tangents. In practice, it works pretty well when the steps are small, but if we need something more accurate there's a whole body of literature around numerical solutions to differential equations of this form. One frequently used method is called Runge Kutta. It has provably better properties, and performs extremely well in practice.

giving us an update of the form

$$\begin{aligned}\boldsymbol{x}_{t+1} &= \boldsymbol{x}_t + \epsilon_t \widetilde{\boldsymbol{v}}(\boldsymbol{x}_t) \\ &= \boldsymbol{x}_t - \epsilon_t \left(\boldsymbol{V}_\perp \boldsymbol{V}_\perp^T\right) \nabla\psi(\boldsymbol{x}_t).\end{aligned}$$

The scalar $\epsilon_t > 0$ should be small enough at each iteration to ensure that the tangent space remains an accurate representation of the surface at the resulting point $\boldsymbol{x}_{t+1}$. Generally, choosing ϵ_t can be a bit of voodoo, but it's often sufficient to use a formula of the form

$$\epsilon_t = \epsilon \ \min\left\{1, \ \frac{1}{\|\boldsymbol{v}\|}\right\}$$

to ensure that the steps are of length at most ϵ, but still shrink to zero as $\boldsymbol{v}$ shrinks to zero. This algorithm will walk across the implicit surface, following the vector field defined by the potential function, until those vectors tell it to stop. A good stopping criterion to check is $\|\widetilde{\boldsymbol{v}}\| \leq \epsilon_{\text{term}}$ for some small $\epsilon_{\text{term}} > 0$. Note that the norm of the unprojected vector $\boldsymbol{v}$ usually doesn't actually approach zero unless the local minimum of the potential function, itself, already rests on the implicit surface. Usually, the projected vector approaches zero as the actual potential vector becomes orthogonal to the tangent space of the surface.

3 An interlude on efficient Pseudoinverse calculations

Section 2.2 builds a simple algorithm for following a potential field across an implicit surface, but the core computation relies on computing the SVD of the Jacobian matrix. The SVD is an exceedingly reliable way to get everything you ever wanted to know about the structure of a matrix and its fundamental spaces in a form you can easily manipulate, but it's sometimes more computationally intensive than problem requirements permit. This section relates both pseudoinverses and null space projectors to convenient and computationally efficient matrix forms that work well when the Jacobian is full rank.

We saw earlier in Ratliff (2014b) that examining the underlying structure of a matrix through its SVD $\boldsymbol{A} = \boldsymbol{U}_{/\!/}\boldsymbol{S}\boldsymbol{V}_{/\!/}^T \in \mathbb{R}^{m\times n}$ exposed a natural pseudoinverse of the form $\boldsymbol{A}^\dagger = \boldsymbol{V}_{/\!/}\boldsymbol{S}^{-1}\boldsymbol{U}_{/\!/}^T$. Here, we explore this expression in more detail and rewrite it in various forms that relate it back to the original matrix and avoid SVD computations. We also derive corresponding matrix forms of the column, row, and left and right null space projectors that build on these pseudoinverse matrix expressions.

First, notice that when the matrix is taller than it is wide and has full rank, its SVD is just $\boldsymbol{A} = \boldsymbol{U}_{/\!/}\boldsymbol{S}\boldsymbol{V}^T$, where $\boldsymbol{V} \in \mathbb{R}^{n^2}$ is a full orthogonal matrix, since there's no (non-trivial) right null space. Thus, we can write $\boldsymbol{A}^T\boldsymbol{A} = \left(\boldsymbol{V}\boldsymbol{S}\boldsymbol{U}_{/\!/}^T\right)\left(\boldsymbol{U}_{/\!/}\boldsymbol{S}\boldsymbol{V}^T\right) = \boldsymbol{V}\boldsymbol{S}^2\boldsymbol{V}^T$. $\boldsymbol{V}$ is orthogonal and $\boldsymbol{S}$ is diagonal with strictly

positive entries, so we can invert this quantity $\left(\boldsymbol{A}^T\boldsymbol{A}\right)^{-1} = \boldsymbol{V}\boldsymbol{S}^{-2}\boldsymbol{V}^T$. Using this relation we can rewrite the pseudoinverse for the full rank case of $m \geq n$ as

$$\begin{aligned}\boldsymbol{A}^\dagger &= \boldsymbol{V}\boldsymbol{S}^{-1}\boldsymbol{U}_{/\!/}^T \\ &= \left(\boldsymbol{V}\boldsymbol{S}^{-2}\boldsymbol{V}^T\right)\boldsymbol{V}\boldsymbol{S}\boldsymbol{U}_{/\!/}^T \quad \text{(Multiplying this out gives the above line.)} \\ &= \left(\boldsymbol{A}^T\boldsymbol{A}\right)^{-1}\boldsymbol{A}^T. \end{aligned} \tag{1}$$

This algebra shows that we can compute the pseudoinverse for full rank matrices with $m \geq n$ using a simple matrix formula. Note that if n is much smaller than m, the matrix $\boldsymbol{A}^T\boldsymbol{A} \in \mathbb{R}^{n^2}$ is small and positive definite so it can easily be inverted.

Likewise, when $m \leq n$ (it's wider than it is tall) and the matrix is full rank, we can write $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}_{/\!/}^T$ with $\boldsymbol{U} \in \mathbb{R}^{m^2}$ a full orthogonal matrix since there's no *left* null space in this case. Now the matrix $\boldsymbol{A}\boldsymbol{A}^T = \boldsymbol{U}\boldsymbol{S}^2\boldsymbol{U}^T$ is invertible with inverse $\left(\boldsymbol{A}\boldsymbol{A}^T\right)^{-1} = \boldsymbol{U}\boldsymbol{S}^{-2}\boldsymbol{U}^T$ and we can write

$$\begin{aligned}\boldsymbol{A}^\dagger &= \boldsymbol{V}_{/\!/}\boldsymbol{S}^{-1}\boldsymbol{U}^T \\ &= \boldsymbol{V}_{/\!/}\boldsymbol{S}\boldsymbol{U}^T\left(\boldsymbol{U}\boldsymbol{S}^{-2}\boldsymbol{U}^T\right) \\ &= \boldsymbol{A}^T\left(\boldsymbol{A}\boldsymbol{A}^T\right)^{-1}. \end{aligned} \tag{2}$$

Again, we obtain an matrix formula for the pseudoinverse that we can used for full rank matrices with $m \leq n$. In this case, the matrix inversion acts on an $m \times m$ dimensional positive definite matrix, so when m is relatively small this operation is pretty efficient.

Equations 1 and 2 give matrix expressions for the pseudoinverse for particular cases $m \geq n$ and $m \leq n$, respectively; we can also build on these expressions to construct matrix forms of the projectors onto the column space and row space. When $m > n$ and the matrix is full rank, we can write

$$\begin{aligned}\boldsymbol{A}\boldsymbol{A}^\dagger &= \boldsymbol{A}\left(\boldsymbol{A}^T\boldsymbol{A}\right)^{-1}\boldsymbol{A}^T \\ &= \boldsymbol{U}_{/\!/}\underbrace{\boldsymbol{S}\boldsymbol{V}^T\left(\boldsymbol{V}\boldsymbol{S}^{-2}\boldsymbol{V}^T\right)\boldsymbol{V}\boldsymbol{S}}_{=\boldsymbol{I}}\boldsymbol{U}_{/\!/} \\ &= \boldsymbol{U}_{/\!/}\boldsymbol{U}_{/\!/}^T, \end{aligned} \tag{3}$$

which gives us a matrix formula for the column space projector. And similarly, it's easy to show that for full rank matrices with $m \leq n$ the matrix $\boldsymbol{A}^\dagger\boldsymbol{A} = \boldsymbol{V}_{/\!/}\boldsymbol{V}_{/\!/}^T$ is the row space projector.

And finally, given that we have matrix expressions for projecting onto the row space and column space, we can easily get expressions for projecting onto the corresponding left and right null spaces by simply subtracting them from

Table 1: Matrix formulas for pseudoinverses and fundamental space projectors.

Condition	Transform	Matrix formula	Name
$m > n$	$\boldsymbol{V}\boldsymbol{S}^{-1}\boldsymbol{U}_{/\!/}^T$	$\left(\boldsymbol{A}^T\boldsymbol{A}\right)^{-1}\boldsymbol{A}^T$	Left pseudoinverse
$m < n$	$\boldsymbol{V}_{/\!/}\boldsymbol{S}^{-1}\boldsymbol{U}^T$	$\boldsymbol{A}^T\left(\boldsymbol{A}\boldsymbol{A}^T\right)^{-1}$	Right pseudoinverse
$m > n$	$\boldsymbol{U}_{/\!/}\boldsymbol{U}_{/\!/}^T$	$\boldsymbol{A}\left(\boldsymbol{A}^T\boldsymbol{A}\right)^{-1}\boldsymbol{A}^T$	Column space projector
$m < n$	$\boldsymbol{V}_{/\!/}\boldsymbol{V}_{/\!/}^T$	$\boldsymbol{A}^T\left(\boldsymbol{A}\boldsymbol{A}^T\right)^{-1}\boldsymbol{A}$	Row space projector
$m > n$	$\boldsymbol{U}_\perp\boldsymbol{U}_\perp^T$	$\boldsymbol{I} - \boldsymbol{A}\left(\boldsymbol{A}^T\boldsymbol{A}\right)^{-1}\boldsymbol{A}^T$	Left annihilator
$m < n$	$\boldsymbol{V}_\perp\boldsymbol{V}_\perp^T$	$\boldsymbol{I} - \boldsymbol{A}^T\left(\boldsymbol{A}\boldsymbol{A}^T\right)^{-1}\boldsymbol{A}$	Right annihilator

the identity.[4] The resulting expressions are

$$\boldsymbol{I} - \boldsymbol{A}\boldsymbol{A}^\dagger = \boldsymbol{I} - \boldsymbol{A}\left(\boldsymbol{A}^T\boldsymbol{A}\right)^{-1}\boldsymbol{A}^T = \boldsymbol{U}_\perp\boldsymbol{U}_\perp^T \quad \text{for } m > n \tag{4}$$

$$\boldsymbol{I} - \boldsymbol{A}^\dagger\boldsymbol{A} = \boldsymbol{I} - \boldsymbol{A}^T\left(\boldsymbol{A}\boldsymbol{A}^T\right)^{-1}\boldsymbol{A} = \boldsymbol{V}_\perp\boldsymbol{V}_\perp^T \quad \text{for } m < n. \tag{5}$$

These expressions are sometimes referred to as *annihilators* since they annihilate the components of a vector lying in the column or row space.

Note that the expression $\boldsymbol{A}^\dagger = \left(\boldsymbol{A}^T\boldsymbol{A}\right)^{-1}\boldsymbol{A}^T$ is used to extract the column space and left null space projector. For that reason, it's often called the *left* pseudoinverse. Likewise, $\boldsymbol{A}^\dagger = \boldsymbol{A}^T\left(\boldsymbol{A}\boldsymbol{A}^T\right)^{-1}$ is used to extract the row space and right null space projector, so it's often called the *left* pseudoinverse. Table 1 summarizes all of these results.

4 The geometry of parameterized surfaces

Above, in Section 2, we saw that one way to represent a surface is implicitly as the zero set of a function. This section explores a second way of representing surfaces: explicit parameterization.

A nice example of such a parameterization is the typical mathematical model for a robot. What is a robot, physically? It's a whole bunch of particles—atoms and molecules of its constituent materials—on the order of 10^{23} of them (Avogadro's number). Entirely unconstrained, the space of all possible configurations which that many particles might be in is huge! The robot is technically a point[5] in $\mathbb{R}^{10^{23}}$. But we already know that it's not really that hard. If we make some assumptions, like the robot's bolted to the ground and is a collection of rigid links connected only pairwise in a chain by revolute joints, we can mathematically describe the configurations of all of those particles using just some small

[4] We can write out this algebra more explicitly for $m \geq n$ by noting that $\boldsymbol{I} = \boldsymbol{U}_{/\!/}\boldsymbol{U}_{/\!/}^T + \boldsymbol{U}_\perp\boldsymbol{U}_\perp^T$; subtracting off the column space projector $\boldsymbol{U}_{/\!/}\boldsymbol{U}_{/\!/}^T$ just leaves the null space projector $\boldsymbol{U}_\perp\boldsymbol{U}_\perp^T$. A similar argument holds for $n \leq m$.

[5] It's not really precisely 10^{23} dimensions, just on the order of that, but for the sake of argument we'll use that number specifically.

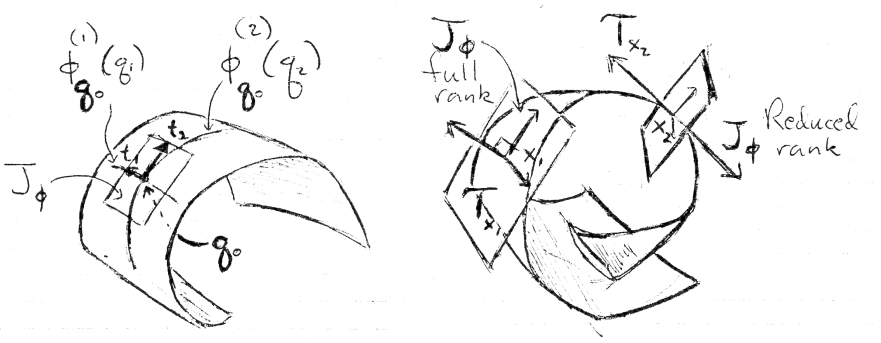

Figure 2: **Left:** A 2D explicitly parameterized surface in 3D defined by $\phi : \mathbb{R}^2 \to \mathbb{R}^3$. **Right:** A mixed dimensionality explicitly parameterized surface in 3D defined by $\phi : \mathbb{R}^2 \to \mathbb{R}^3$ with a region around the middle where it reduces rank.

set of numbers: the joint angles. Denoting those joint angles as $\boldsymbol{q} \in \mathbb{R}^d$ this suggests we can write down a mapping of the form $\phi : \mathbb{R}^d \to \mathbb{R}^{10^{23}}$ that tells us tells us the configuration of all of those particles for a specific setting of the joints.

We know such a mapping to be differentiable and we often say that this sort of robot is of dimension d. But why is that? What we'll see in this section is that this mapping defines a smooth d-dimensional (almost everywhere) surface embedded in an ambient 10^{23} dimensional space. Clearly, not all configurations of the particles are possible—the robot doesn't typically spontaneously evaporate into a mist of freely moving constituent particles—instead the simultaneous configurations of the particles are severely constrained, and specifically lie only on this lower-dimensional surface.

4.1 Again, the Jacobian tells us everything

Lets analyze the generic case of such a surface to better understand what the Jacobian of the linear map can tell us about its structure. Suppose $\phi : \mathbb{R}^d \to \mathbb{R}^n$ is a d-dimensional map into an n-dimensional ambient space. At any given point $\boldsymbol{q}_0 \in \mathbb{R}^d$, we can look how the map behaves when we locked all dimensions except the jth one and just vary that one dimension. The resulting map $\phi^{(j)}_{\boldsymbol{q}_0}(q_j)$ from that one-dimension to the n-dimensional ambient space defines a *trajectory* $\phi^{(j)}_{\boldsymbol{q}_0} : \mathbb{R} \to \mathbb{R}^n$ across the surface which intersects the point $\phi(\boldsymbol{q}_0)$. If we look at all of these trajectories together for $j = 1, \ldots, d$, we see they form a local *curvilinear coordinate system* across the surface centered at $\phi(\boldsymbol{q}_0)$, as pictured in Figure 2. More importantly, the set of all tangents $\boldsymbol{t}_j = \frac{\partial \phi^{(j)}_{\boldsymbol{q}_0}}{\partial q_j}$ to the trajectories at $\boldsymbol{q}_0$ span a linear space tangent to the surface at $\boldsymbol{x}_0 = \phi(\boldsymbol{q}_0)$. This linear space $\mathbf{span}\{\boldsymbol{t}_i\}_{i=1}^d$ is precisely the *tangent space* to the surface at that point (which tells us more explicitly why the space is called a "tangent space"). Collecting

all these tangent vectors up as columns of a matrix, we see that the resulting matrix is just the Jacobian

$$\begin{bmatrix} | & | & & | \\ \boldsymbol{t}_1 & \boldsymbol{t}_2 & \cdots & \boldsymbol{t}_d \\ | & | & & | \end{bmatrix} = \begin{bmatrix} | & | & & | \\ \frac{\partial \phi^{(1)}_{\boldsymbol{q}_0}}{\partial q_1} & \frac{\partial \phi^{(2)}_{\boldsymbol{q}_0}}{\partial q_2} & \cdots & \frac{\partial \phi^{(d)}_{\boldsymbol{q}_0}}{\partial q_d} \\ | & | & & | \end{bmatrix} = \frac{\partial \phi}{\partial \boldsymbol{q}} = \boldsymbol{J}_\phi. \quad (6)$$

In other words, the columns of the Jacobian at a given point $\boldsymbol{q}_0$ span the tangent space $\mathcal{T}_{\boldsymbol{q}_0}$ at that point: the Jacobian's column space *is* the tangent space.

Notice that this situation is the opposite of what we saw for the implicit surface representation. There, the surface is the zero set, so the tangent space is the space of all directions that don't, to the first-order, change the map's value. So, in that case, the tangent space was the Jacobian's *null* space. In this section, the map explicitly describes the surface and each input dimension defines a trajectory through the surface. For explicitly parameterized surfaces of this form the tangent space at a point is explicitly the space of all tangents to trajectories passing through that point. That space is spanned by the tangents to the d specific *coordinate* trajectories defined above. And we see in Equation 6 that that space is exactly the *column* space of the map's Jacobian.

Finally, we note that, since the tangent space and the Jacobian's column space are one and the same for parametric surfaces, reduced rank Jacobians change the dimensionality of the space. The rightmost subplot of Figure 2 shows an instance of a mostly two-dimensional parameterized surface which drops rank toward the middle reducing it to a one-dimensional surface within that region. A point of reduced dimensionality is called a **singularity**. It's signified by a reduction of rank in the nonlinear map's Jacobian.

4.2 Following potential fields across parameterized surfaces

We can again ask the question of how we might best follow a vector field induced by a potential function in the ambient space while sticking to the surface. In this case, if we walk through the space of inputs $\boldsymbol{q}$ to the map, there's no possibility of ever falling off the surface into the surrounding ambient space, but now the question is how do we step in the right direction?

Looking at the problem from the ambient space as we did in Section 2.2, we can say geometrically that if we want to move in the direction $\boldsymbol{v} = -\nabla\psi(\boldsymbol{x})$ at a given point, we should project the vector onto the tangent space and step in that direction. Noting that the tangent space is the Jacobian's column space and decomposing the Jacobian in terms of its SVD as $\boldsymbol{J}_\phi = \boldsymbol{U}_{/\!/}\boldsymbol{S}\boldsymbol{V}_{/\!/}^T$, this logic suggests we should move in the direction $\widetilde{\boldsymbol{v}} = \left(\boldsymbol{U}_{/\!/}\boldsymbol{U}_{/\!/}^T\right)\boldsymbol{v}$.

But that's not enough for this problem. We aren't stepping through the ambient space, we're stepping through the space of parameters $\boldsymbol{q}$. So we need to calculate what direction in $\boldsymbol{q}$ corresponds to this projected step $\widetilde{\boldsymbol{v}}$ along the tangent space. To understand the geometry of the mapping connecting the

space of parameters to the ambient space, we can again take the first-Taylor approximation at our current point $\boldsymbol{q}_0$ and rewrite it

$$\underbrace{\phi(\boldsymbol{q}) - \phi(\boldsymbol{q}_0)}_{\delta \boldsymbol{x}} \approx \boldsymbol{J}_\phi \underbrace{(\boldsymbol{q} - \boldsymbol{q}_0)}_{\delta \boldsymbol{q}}, \tag{7}$$

giving us the simple relation $\delta \boldsymbol{x} \approx \boldsymbol{J}_\phi \delta \boldsymbol{q}$, where the approximation gets more and more accurate as $\delta \boldsymbol{q}$ becomes increasingly small. So now our question reduces specifically to what perturbation $\delta \boldsymbol{q}$ corresponds to an ambient motion $\delta \boldsymbol{x} = \widetilde{\boldsymbol{v}}$?

We've already studied this problem quite a bit in Ratliff (2014b). We know that $\widetilde{\boldsymbol{v}}$ lies in the column space, so the corresponding input vector is just $\delta \boldsymbol{q}^* = \boldsymbol{J}_\phi^\dagger \widetilde{\boldsymbol{v}} = \boldsymbol{V}_{/\!/} \boldsymbol{S}^{-1} \boldsymbol{U}_{/\!/}^T \widetilde{\boldsymbol{v}}$. Notice that, expanding $\widetilde{\boldsymbol{v}} = \left(\boldsymbol{U}_{/\!/} \boldsymbol{U}_{/\!/}^T \right) \boldsymbol{v}$, we see that

$$\begin{aligned} \delta \boldsymbol{q}^* &= \boldsymbol{V}_{/\!/} \boldsymbol{S}^{-1} \boldsymbol{U}_{/\!/}^T \left(\boldsymbol{U}_{/\!/} \boldsymbol{U}_{/\!/}^T \right) \boldsymbol{v} \\ &= \boldsymbol{V}_{/\!/} \boldsymbol{S}^{-1} \boldsymbol{U}_{/\!/}^T \boldsymbol{v}. \end{aligned}$$

In retrospect, we already knew that. We projected the ambient vector $\boldsymbol{v}$ onto the tangent space before pulling it back through the pseudoinverse to the space of parameters. But we know that the pseudoinverses actually does this automatically—they naturally do the best they can when a map isn't invertible, which, in this case, is to first project onto the column space, and then pull the resulting vector back to the row space. The above observations are just a manifestation of that property.

So we arrive at another algorithm for walking across a surface in an ambient space following to the best of our ability the vector directions of a vector field induces by a potential function ψ. In this case, the algorithm suggests we calculate the negative gradient potential field direction $\boldsymbol{v}_t = -\nabla_{\boldsymbol{x}} \psi(\boldsymbol{x}_t)$ at our current ambient point $\boldsymbol{x}_t = \phi(\boldsymbol{q}_t)$, and take a step of the form

$$\begin{aligned} \boldsymbol{q}_{t+1} &= \boldsymbol{q}_t + \epsilon_t \boldsymbol{J}_\phi^\dagger \boldsymbol{v}_t \\ &= \boldsymbol{q}_t - \epsilon_t \left(\boldsymbol{J}_\phi^T \boldsymbol{J}_\phi \right)^{-1} \boldsymbol{J}_\phi^T \nabla_x \psi(\boldsymbol{x}_t), \end{aligned} \tag{8}$$

where we've written out the left pseudoinverse in its matrix form assuming the Jacobian is full rank. Again $\epsilon_t > 0$ is some step size small enough to ensure that the resulting point $\boldsymbol{q}_{t+1}$ is close enough to $\boldsymbol{q}_t$ that the linear approximations remain good representations of the map.

This algorithm is geometrically motivated using the same arguments we used to walk across the implicit surface in Section 2.2. Indeed, this method should (approximately, as long as the steps are small) engender the same behavior as the algorithm we derived in that section if we have an equivalent implicit surface representation.

This point touches on a notion of what we call **covariance**, which is a deep concept of fundamental importance to physics, and one cited frequently in fields related to intelligent systems. The algorithm we derived was, in a basic sense, purely geometric. Our intuition built on the ideal of taking steps across the

surface through the ambient space, and only then did we figured out what the updates would need to be to implement those steps using both an implicit surface representation and an explicit parametric surface representation. Importantly, although the underlying representation changes between the two algorithms, the underlying behavior of the steps through the ambient space doesn't.

This invariance to representation is an ideal property for algorithms designed for systems, such as physical systems in the real world, that have some notion of underlying existence independent of their representation. For instance, in robotics, the behavior of control and planning algorithms for physical robots should depend on the physical properties of the system we're modeling, not on an arbitrary choice of numbers used to represent the system on a computer. Similarly, in machine learning if we're manipulating a function, the algorithm's behavior should depend on basic properties of the function space itself (such as its inner product as defined by a kernel in a Reproducing Kernel Hilbert Space Schölkopf & Smola. (2002)), and not on how we specifically choose to parameterize the functions. We can't always achieve covariance, but it's something we strive for. And when we do, what results are often very robust, versatile, and reliable algorithms.

5 On steepest descent and metric dependence

Notice that the right thing to do in the previous section was *not* to take steps in the direction of the usual gradient of the nested function $\widetilde{\psi}(\boldsymbol{q}) = \psi(\phi(\boldsymbol{q}))$. Such a step would be of the form

$$\boldsymbol{q}_{t+1} = \boldsymbol{q}_t - \epsilon_t \boldsymbol{J}_\phi^T \nabla_{\boldsymbol{x}} \psi(\boldsymbol{x}_t).$$

Instead, we take that *parametric*[6] gradient and transform it through the inverse of a matrix $\boldsymbol{A}(\boldsymbol{q}) = \boldsymbol{J}_\phi^T \boldsymbol{J}_\phi$ before taking the step. Why is that? Didn't we say previously in (Ratliff, 2014c) that the gradient is the direction of fastest increase of the function? (Or, equivalently, in the opposite direction, fastest decrease?)

Let's examine that statement more closely. Remember from the first-order Taylor expansion, we see that a function in $\boldsymbol{q}$ may be written approximately as

$$\widetilde{\psi}(\boldsymbol{q}) \approx \widetilde{\psi}(\boldsymbol{q}_0) + \nabla\widetilde{\psi}(\boldsymbol{q}_0)^T \delta\boldsymbol{q},$$

where $\delta\boldsymbol{q}$ is a small perturbation away from $\boldsymbol{q}_0$ so that $\boldsymbol{q} = \boldsymbol{q}_0 + \delta\boldsymbol{q}$. The notion that the gradient is the "most efficient" direction of increase is that if we restrict $\delta\boldsymbol{q}$ to be of a particular small norm $\|\delta\boldsymbol{q}\| = \epsilon$, then any component orthogonal to $\nabla\widetilde{\psi}$ is "wasted" in the sense that it doesn't contribute to the inner product, but still reduces the size of the component that *does* align with $\nabla\widetilde{\psi}$ and contribute to the inner product. The overall inner product is then necessarily smaller than

[6] It's the parametric gradient in the sense that $\boldsymbol{x} = \phi(\boldsymbol{q})$ can often be viewed as creating an explicity parameterized surface in an ambient space as we discuss in Section 4. This naïve gradient is the gradient with respect to simply these parameters, not with respect to any natural underlying geometry of the ambient space.

it could have been if we had made the *entire* perturbation vector align with the gradient. So the direction that increases the function the fastest, *if we force the vector to be at most size* ϵ, is the direction that fully aligns with the gradient $\nabla\widetilde{\psi}$.

That argument entirely revolves around the idea that we're restricting the *Euclidean* norm of the perturbation vector $\delta\boldsymbol{q}$ to be of length ϵ. What if we don't care about Euclidean norms? What if we have some positive definite matrix $\boldsymbol{A}$ and we care about measuring the size of perturbations in $\boldsymbol{q}$ with respect to the norm $\|\delta\boldsymbol{q}\|_{\boldsymbol{A}}^2 = \delta\boldsymbol{q}^T\boldsymbol{A}\delta\boldsymbol{q}$. How does that change the argument?

Lets do brute force calculation of the **direction of steepest descent** to fully analyze how this new norm affects the direction. Strictly speaking we need to find a vector $\delta\boldsymbol{q}$ whose norm $\|\delta\boldsymbol{q}\|_{\boldsymbol{A}}$ is never greater than ϵ and whose value $\widetilde{\psi}(\boldsymbol{q}_0 + \delta\boldsymbol{q})$ is as small as possible. This is a nonlinear constrained optimization problem, so in general that's hard. But since ϵ is small we can simplify it by taking the first-order Taylor approximation. Additionally, we won't prove it here, but the resulting constrained problem is equivalent to the following unconstrained quadratic function minimization problem

$$\min_{\delta\boldsymbol{q}} \quad \widetilde{\psi}(\boldsymbol{q}_0) + \nabla\widetilde{\psi}(\boldsymbol{q}_0)^T\delta\boldsymbol{q} + \frac{\lambda}{2}\|\delta\boldsymbol{q}\|_{\boldsymbol{A}}^2,$$

for some value of $\lambda > 0$ (which is inversely related to the choice of ϵ). The first two terms form the first-order Taylor expansion, and the last term ensures that $\delta\boldsymbol{q}$ stays small, where we specifically define this notion of "small" with respect to the measure of interest.

This problem is just a quadratic minimization, so we can solve it analytically by setting its gradient with respect to $\delta\boldsymbol{q}$ to zero. Doing so gives

$$\begin{aligned} \nabla\widetilde{\psi}(\boldsymbol{q}_0) + \lambda\boldsymbol{A}\,\delta\boldsymbol{q} &= \boldsymbol{0} \\ \Rightarrow \delta\boldsymbol{q} &= -\frac{1}{\lambda}\boldsymbol{A}^{-1}\,\nabla\widetilde{\psi}(\boldsymbol{q}_0). \end{aligned}$$

In other words, if we really do care about the Euclidean norm, the direction of steepest descent is precisely the gradient since $\boldsymbol{A} = \boldsymbol{I}$. But if we have some non-trivial positive definite matrix $\boldsymbol{A} \neq \boldsymbol{I}$ that more naturally measures the size of perturbations in this space, the *right* thing to do is to transform that gradient by $\boldsymbol{A}^{-1}$. The negative transformed gradient is the direction of steepest descent under that metric, not the gradient itself.

This transformed gradient is sometimes known as the **natural gradient** with respect to the **Riemannian metric** $\boldsymbol{A}$ (Lee, 2002; Hassani, 2013; Amari & Nagaoka, 1994). In general, a Riemannian metric may change smoothly from point to point in the space. Technically, it forms an inner product on the tangent space of a manifold, but for our purposes, we can think of it as simply defining a norm $\|\delta\boldsymbol{q}\|_{\boldsymbol{A}(\boldsymbol{q})}$ on perturbation vectors $\delta\boldsymbol{q}$ that move away from the point $\boldsymbol{q}$. As we've seen, this particular choice of norm can have a profound effect on the direction of steepest descent!

Looking back at Equation 8, we can now see why we weren't taking a step directly in the direction of the gradient $\nabla\widetilde{\psi}(\boldsymbol{q}) = \boldsymbol{J}_{\phi}^T\nabla_{\boldsymbol{x}}\psi(\phi(\boldsymbol{q}))$. That gradient

isn't the direction of steepest descent with respect to the norm we care about. By consider geometry in the ambient space and performing orthogonal projections in that space, we're implicitly defining the size of perturbations with respect to how they're measured in the *ambient* space. As suggested at the beginning of this section, the metric that results from defining norms that way is $\boldsymbol{A}(\boldsymbol{q}) = \boldsymbol{J}_\phi^T \boldsymbol{J}_\phi$, which is a non-constant Riemannian metric defined for all $\boldsymbol{q}$. The next section describes in more detail how that metric arises and more generally how metrics of that sort arise for entire chains of smooth maps or even tree-structured collections of interconnected maps.

6 What metric should we use?

Natural gradients can be a convenient way to get covariant algorithmic behavior that's agnostic to representation. But what metric should we use if we're just handed some obscure collection of interconnected smooth maps? Even if we know what geometry we really care about in one of the spaces how do we effectively track how that geometry changes from space to space through these nonlinear transformations? Again, the answer lies in the Jacobians.

In Section 4.2, we had a situation where our smooth map $\phi(\boldsymbol{q}) = \boldsymbol{x}$ linked a parameter space $\boldsymbol{q}$ to an ambient space $\boldsymbol{x}$. We decided that we cared specifically about the geometry of the ambient space (we chose to calculate orthogonal projections in that space), so effectively, we placed a metric $\boldsymbol{B} = \boldsymbol{I}$ in the ambient space of $\boldsymbol{x}$. From the first-order Taylor expansion around a point of interested, we see that small perturbations of the parameters $\delta\boldsymbol{q}$ are linked to small perturbations $\delta\boldsymbol{x}$ within the tangent space of the surface as $\delta\boldsymbol{x} = \boldsymbol{J}_\phi \delta\boldsymbol{q}$. Thus, we can simply rewrite the expression $\|\delta\boldsymbol{x}\|^2_{\boldsymbol{B}}$ in terms of $\boldsymbol{q}$:

$$\|\delta\boldsymbol{x}\|^2_{\boldsymbol{B}} \;=\; \delta\boldsymbol{x}^T \boldsymbol{B} \delta\boldsymbol{x} \;=\; \delta\boldsymbol{q}^T \underbrace{\left(\boldsymbol{J}_\phi^T \boldsymbol{B} \boldsymbol{J}_\phi \right)}_{\boldsymbol{A}(\boldsymbol{q})} \delta\boldsymbol{q}.$$

Looking back at Equation 8 we see that that's precisely the metric we're using (with $\boldsymbol{B} = \boldsymbol{I}$). The step through the parameter space we derived geometrically is the steepest descent direction (negative natural gradient) calculated with respect to the metric $\boldsymbol{J}_\phi^T \boldsymbol{J}_\phi$.

This process of sandwiching the metric between the Jacobian and the transposed Jacobian of a nonlinear map connecting spaces *pulls* the metric $\boldsymbol{B}$ back from the co-domain through the nonlinear map ϕ and into the domain. The resulting metric $\boldsymbol{A}(\boldsymbol{q}) = \boldsymbol{J}_\phi^T \boldsymbol{B}(\boldsymbol{x}) \boldsymbol{J}_\phi$ in the domain, where $\boldsymbol{x} = \phi(\boldsymbol{q})$, is called the **pullback metric** of $\boldsymbol{B}$ through ϕ.

The pullback metric is the metric that measures perturbations in the domain the same way that the corresponding perturbations in the co-domain are measured. In this sense, the pullback transfers the geometry of one space (the co-domain) to another (the domain) without altering the basic first-order quality of the geometry. When two spaces are linked through the nonlinear map

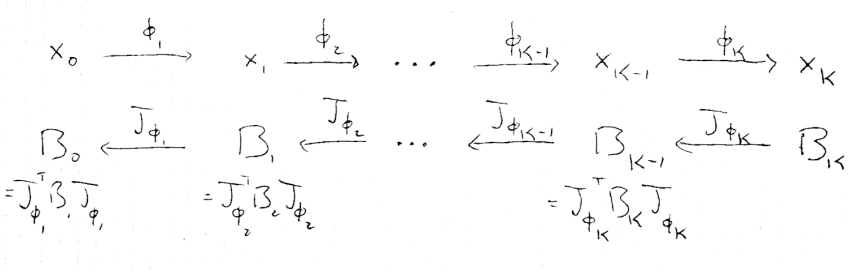

Figure 3: A diagram depicting how Riemannian metrics propagate back through chained nonlinear maps $\phi_K \circ \phi_{K-1} \circ \cdots \circ \phi_1$ from the final co-domain of ϕ_K to the initial domain of ϕ_1 by sequentially being pulled back through each map in turn from the last ϕ_K to the first ϕ_1.

$\boldsymbol{x} = \phi(\boldsymbol{q})$, designing algorithms in $\boldsymbol{x}$ under metric $\boldsymbol{B}(\boldsymbol{x})$ is equivalent to designing algorithm in $\boldsymbol{q}$ under the metric $\boldsymbol{A}(\boldsymbol{q}) = \boldsymbol{J}_\phi^T \boldsymbol{B}(\boldsymbol{x}) \boldsymbol{J}_\phi$ (to the first order).

6.1 Chaining nonlinear maps

Generalizing this idea even further, we see in this section that the chain rule gives a convenient corresponding rule for pulling metrics back through chains of nested (composed) maps. Suppose ϕ_1 maps into the domain of ϕ_2, which maps into the domain of ϕ_3, and so on through K different maps. In combination we can write

$$\phi_{\text{tot}} = \phi_K \circ \cdots \circ \phi_2 \circ \phi_1.$$

From the above analysis we know that if $\boldsymbol{B}$ is the metric defined in the co-domain of the complete map ϕ_{tot}, it's pullback into the domain (which is also the domain of ϕ_1) is $\boldsymbol{A} = \boldsymbol{J}_{\phi_{\text{tot}}}^T \boldsymbol{B} \boldsymbol{J}_{\phi_{\text{tot}}}$. But we can expand the total Jacobian into a product of constituent Jacobians by invoking the chain rule $\boldsymbol{J}_{\phi_{\text{tot}}} = \boldsymbol{J}_{\phi_K} \cdots \boldsymbol{J}_{\phi_2} \boldsymbol{J}_{\phi_1}$. This means the pullback can be rewritten as a nested collection of single map pullbacks

$$\boldsymbol{A} = \boldsymbol{J}_{\phi_1}^T \left(\cdots \left(\boldsymbol{J}_{\phi_{K-1}}^T \left(\boldsymbol{J}_{\phi_K}^T \boldsymbol{B} \boldsymbol{J}_{\phi_K} \right) \boldsymbol{J}_{\phi_{K-1}} \right) \cdots \right) \boldsymbol{J}_{\phi_1}. \tag{9}$$

In other words, the total pullback across the entire sequence of composed maps is precisely the metric you get by pulling $\boldsymbol{B}$ sequentially back through each map in turn, one-by-one, starting from ϕ_K and proceeding through ϕ_{K-1} and so on down all the way to ϕ_1. Figure 3 depicts this process for a single chain of maps.

Note that it's often the case that the metric on the input space isn't fully subservient to the metric in the output space as we've been making it. For instance, if we have a nonlinear map $\boldsymbol{x} = \phi(\boldsymbol{q})$, we might want to measure the size of system perturbations mostly with respect to how they manifest of the

output space $\delta\boldsymbol{x}$, but we might also care some about how they manifest in the input space $\delta\boldsymbol{q}$. As an example, suppose ϕ is a forward kinematics map of a robot manipulator, the space of $\boldsymbol{q}$ is actually a space of joint angles which have a physical manifestation in terms of the robot's mechanics. It might make sense then to place a metric $\boldsymbol{C}$ in that space and metric $\boldsymbol{B}$ in the end-effector space. In combination, the size of a movement is then defined jointly between $\boldsymbol{q}$ and $\boldsymbol{x}$ as

$$\|\delta\boldsymbol{x}\|^2_{\boldsymbol{B}} + \|\delta\boldsymbol{q}\|^2_{\boldsymbol{C}} = \delta\boldsymbol{q}^T \left(\boldsymbol{J}_\phi^T \boldsymbol{B} \boldsymbol{J}_\phi + \boldsymbol{C}\right) \delta\boldsymbol{q}.$$

This expression leverages both representations $\boldsymbol{q}$ and $\boldsymbol{x}$ of the single underlying system simultaneously to define the norm, and then uses the first-order Taylor expansion of the connecting map $\boldsymbol{x} = \phi(\boldsymbol{q})$ to express it in solely in terms of the input space. The resulting metric is the sum[7] of the innate metric in $\boldsymbol{q}$ and the pulled back metric from $\boldsymbol{x}$. These ideas generalize naturally to more complex tree-structured networks of smooth maps where there may be multiple maps defined on the same domain linking it to multiple different co-domains each with its own underlying Riemannian metric. We explore this generalization next.

6.2 Tree structured networks of maps

Combining this idea of defining the metric across all manifestations of the underlying space with the above notion of chaining nonlinear maps, we can see that we can derive metrics for entire tree-structured networks of maps. For instance, suppose we have some underlying space $\boldsymbol{q}$ and a collection of linked spaces $\boldsymbol{x}_i = \phi_i(\boldsymbol{q})$ for $i = 1, \ldots, N$. And suppose each of those had M additional linked spaces $\boldsymbol{y}_{ij} = \psi_{ij}(\boldsymbol{x}_i)$ for $j = 1, \ldots, M$. Then we can measure the size of a perturbation to this system jointly across all variables as

$$\|\delta\boldsymbol{q}\|^2_{\boldsymbol{A}} = \sum_{i=1}^{N} \left(\|\delta\boldsymbol{x}_i\|^2_{\boldsymbol{A}_i} + \sum_{j=1}^{M} \|\delta\boldsymbol{y}_{ij}\|^2_{\boldsymbol{B}_{ij}} \right) + \|\delta\boldsymbol{q}\|^2_{\boldsymbol{C}}, \tag{10}$$

where on the left we denote that the entire metric can be represented entirely in $\boldsymbol{q}$ since the tree-structure of the network of smooth maps is rooted at $\boldsymbol{q}$. The total metric $\boldsymbol{A}$, then is

$$\boldsymbol{A} = \sum_{i=1}^{N} \boldsymbol{J}_{\phi_i}^T \left(\boldsymbol{A}_i + \sum_{j=1}^{M} \boldsymbol{J}_{\psi_{ij}}^T \boldsymbol{B}_{ij} \boldsymbol{J}_{\psi_{ij}} \right) \boldsymbol{J}_{\phi_i} + \boldsymbol{C}. \tag{11}$$

An example of such tree-structured metric is found in the robotics example given above in Section 4. There we saw that a robot may be viewed as a

[7]Note that sometimes the metrics the two spaces are defined independent of their use in combination, so you'll see scaling factors on one or both of the terms. In our case, we assume that such factors are folded into the constituent metrics, themselves.

mapping from its configuration space $\boldsymbol{q}$ to the collection of all $N \sim O(10^{23})$ particles making up its structure

$$\phi_{rob}(\boldsymbol{q}) = \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{x}_N \end{bmatrix}.$$

We often simplify this map by removing most of the output dimensions and focusing on only a handful of useful dimensions (such as the end-effector location) but suppose, for the sake of this argument, that we have access to the entire map and we don't care about computational tractability. The instantaneous kinetic energy of any individual particle on the robot is $\mathcal{K}(\boldsymbol{x}_i) = \frac{1}{2} m_i \|\dot{\boldsymbol{x}}_i\|^2$ (half the particle's mass m_i times its velocity). These velocities are little perturbations in the variable $\boldsymbol{x}_i$, and they're linked to corresponding velocities in $\boldsymbol{q}$ by the equation $\dot{\boldsymbol{x}}_i = \boldsymbol{J}_{\boldsymbol{x}_i} \dot{\boldsymbol{q}}$ where $\boldsymbol{J}_{\boldsymbol{x}_i}$ is the portion of the full Jacobian corresponding to only $\boldsymbol{x}_i$. The total kinetic energy of the system is then

$$\begin{aligned} \mathcal{K}(\boldsymbol{q}, \dot{\boldsymbol{q}}) &= \frac{1}{2} \sum_{i=1}^{N} m_i \|\dot{\boldsymbol{x}}_i\|^2 \\ &= \frac{1}{2} \dot{\boldsymbol{q}}^T \underbrace{\left(\sum_{i=1}^{N} m_i \boldsymbol{J}_{\boldsymbol{x}_i}^T \boldsymbol{J}_{\boldsymbol{x}_i} \right)}_{\boldsymbol{M}(\boldsymbol{q})} \dot{\boldsymbol{q}}. \end{aligned}$$

This matrix $\boldsymbol{M}(\boldsymbol{q})$ plays a critical role in robot control[8] as it expresses a generalized notion of mass (or inertia) as described in $\boldsymbol{q}$. However, we know from the above discussion that we can also view it as a Riemannian metric. This metric measures the size of motion through the configuration space in terms of its kinetic energy.

7 Closing comments

In light of all these observations, we see it's important to acknowledge that in many cases, especially when physical systems are involved, the simple Euclidean norm in a space is often arbitrary and largely meaningless. You should be skeptical of orthogonal projections in spaces of parameters that don't account for a more holistic geometry of the system. When the problem is understood from a representation independent viewpoint we can analyze the algorithmic behavior

[8] Note that typically we model the robot as a continuum of particles across its links, so sums become integrals. In those cases, we can typically simplify the calculation since the mass of a rigid link can be represented (for the sake of kinetic energy) by its total mass and rotational inertial around its centroid. Those ideas are the basis of rigid body dynamics calculations which are central to many areas of modern robotics, but for our purposes here we can think of it as simply a sum across all of the many many point particles constituting the robot.

of the underlying system and not just the behavior of an arbitrary implementation detail. This theme is strongly related to the motivation behind defining vector spaces and their underlying inner products abstractly (Ratliff, 2014a). Indeed, the study of Riemannian geometry is usually much more abstract than the cursory presentation we gave above. Rigorous developments leverage many of these same ideas and tools seen in abstract constructions of linear algebra (Lee, 2002).

Fortunately, we can understand the basic nature of how geometry transfers across smooth maps by relating it to linear algebra using the Jacobian and first-order Taylor expansion, and in doing so we gain a powerful new tool set for algorithmic design and analysis. Some of these results can also be built from a simple manipulations of quadratic functions. For instance, a lot of control algorithms are motivated this way—the resulting quadratic's Hessian plays a role very similar to the Riemannian metric discussed above—but doing so can obscure the underlying geometry of the problem. Even when building algorithms using least squares and quadratic optimization tools it can be insightful to think through the behavior of the resulting algorithm in terms of the geometric arguments presented in this document.

For further, in depth, study of these ideas, it's best to start with a strong foundation of Topology and progress through smooth manifolds to the final generalization of Riemannian geometry. Some of the underlying mathematics of machinery gets pretty complicated, but often a good introduction emphasizing intuition over pure mathematical proof can be found in physics texts. For instance, Hassani (2013) is an excellent book on Mathematical physics, although that presentation of Riemannian geometry requires a lot of background reading (the book is self-contained, but really big). Another good monograph on the subject is Isham (1999). For a more mathematically complete, but very well written, treatment, see Lee (2011, 2002, 1997). These ideas are becoming increasingly important in robotics and especially machine learning (Amari & Nagaoka, 1994) and optimization (Absil et al., 2008), so a little exploration and investment into understanding the underlying theory can go a long way.

References

Absil, P.-A., Mahony, R., and Sepulchre, R. *Optimization Algorithms on Matrix Manifolds*. Princeton University Press, 2008.

Amari, Shun-Ichi and Nagaoka, Hiroshi. *Methods of Information Geometry*. American Mathematical Society, 1994. Translated 2000, and renewed 2007.

Hassani, Sadri. *Mathematical Physics: A Modern Introduction to its Foundations*. Springer, 2nd edition, 2013.

Isham, Chris J. *Modern Differential Geometry for Physicists*. World Scientific Publishing Co. Re. Ltd., 2nd edition, 1999.

Lee, John M. *Riemannian Manifolds: An Introduction To Curvature*. Springer, 1997.

Lee, John M. *Introduction to Smooth Manifolds*. Springer, 2nd edition, 2002.

Lee, John M. *Introduction to Topological Manifolds*. Springer, 2nd edition, 2011.

Ratliff, Nathan. Linear algebra II: The fundamental structure of linear transforms, 2014a. Lecture notes: Mathematics for Intelligent Systems series.

Ratliff, Nathan. Linear algebra III: Computations in coordinates, 2014b. Lecture notes: Mathematics for Intelligent Systems series.

Ratliff, Nathan. Multivariate calculus I: Derivatives and local geometry, 2014c. Lecture notes: Mathematics for Intelligent Systems series.

Schölkopf, Bernhard and Smola., Alex J. *Learning with Kernels*. MIT Press, 2002.