First Order Methods on Matrix Manifolds

Course Project For CSCI-GA.2332
Convex & Non-Smooth Optimization

James Bannon

May 14, 2018

Abstract

This project is a summary of the theory and algorithms of steepest descent on Riemannian matrix manifolds. Matrix manifolds are special cases of manifolds where the elements either *are* matrices or can be represented as matrices. These have the advantage of making the abstract formulations of Riemannian geometry amenable to numerical computation. This suggests an *optimization framework* that follows a simple program for solving constrained optimization problems. First, give the feasible set defined by the constraints the form of a Riemannian manifold. Second, perform *unconstrained* minimization of the objective function on that manifold. The solution will be both feasible — as it will be on the manifold — and optimal — assuming convergence. Thus this framework can be seen as a way of generalizing the familiar unconstrained methods from $\mathbb{R}^n$ that allows for problems not usually addressable by descent methods to be attacked with these approaches.

To investigate this topic I read the first four chapters of *Optimization Algorithms on Matrix Manifolds* by P.A. Absil, R. Mahony, and R. Sepulchre [2] as well as the appendices. This project contains a summary of the topics in those chapters with additional pointers towards more advanced theory — such as second order & trust-region methods on manifolds — as well as some applications of the methods.

The content of this report roughly mirrors that of those chapters. It includes an introduction to Riemannian geometry with focus on developing the necessary theory for the statement of an analogue of gradient descent on Riemannian manifolds. This leads naturally to a discussion of the convergence analysis of the method, which is followed by examples of applying to the theory to three distinct problems with two numerical experiments. Most of the applications given in the text [2] are about finding invariant subspaces, in particular for finding eigenvalues and eigenvectors. We recreate one such example here but then apply the techniques to new problems.

1 Introduction

Descent methods have many advantages. They are intuitive, easy to implement, and in many settings they have nice convergence guarantees. Descent methods follow a simple program to develop a sequence of iterates[1] $\{x^{(i)}\}_{i=1}^{\ell}$ by repeatedly choosing a *step direction* $\Delta x^{(k)}$ and a *step size* $t^{(k)}$ and choosing the next iterate to be $x^{(k+1)} := x^{(k)} + t^{(k)}\Delta x^{(k)}$. The most commonly used descent method is *gradient descent* or *steepest descent* which sets the step direction to be $-\nabla f(x^{(k)})$, the negative gradient of f, the objective function, at the iterate $x^{(k)}$. Pseudocode for this method is given in Algorithm 1.

In a constrained optimization setting pure descent methods will need to be altered, for example by adding a projection step to return to the feasible set after a descent step is taken. This approach has its limits because the projection may not be easily computable and so may degrade performance, or it may not even *be* computable. These restrictions are one of the reasons why other methods — such as primal-dual interior points methods or barrier methods (as described in[4])— that must *stay* feasible were developed and are widely used for constrained problems.

[1]For theoretical analyses we assume the sequence is infinite but for practical purposes the algorithm returns a finite sequence.

This project is about one way of circumventing these difficulties. In particular it is about ways of re-framing constrained optimization problems with smooth costs such that they can be solved with a generalized version of gradient descent.

Speaking roughly the program proposed in [2] and summarized here is an *optimization framework* that operates along the following lines. Suppose we have a generic optimization problem:

$$
\begin{aligned}
\underset{x\in\mathbb{R}^n}{\text{minimize}} \quad & f_0(x) \\
\text{subject to} \quad & f_j(x) \le 0 \; j = 1,\dots,k \\
& h_i(x) = b_i \; i = 1,\dots,m
\end{aligned}
\tag{1}
$$

where we make no assumptions about the convexity of neither the objective function $f_0(x)$ nor the constraints, but assume f_0 is smooth. The feasible set can be seen as a subset of some ambient space. For example in the case of the *least squares* problem where $f_0(x) = ||Ax - b||_2^2$ without constraints, the feasible set is the total space $\mathbb{R}^n$. In the case of the constrained (sometimes called *Ivanov*) form of the lasso — where f_0 is the same as the least squares case and we have one constraint $f_1(x) = ||x||_1 - \lambda \le 0$ for $\lambda > 0$ — the feasible set is a polyhedral "ball" around the origin in $\mathbb{R}^n$. In the case of semidefinite programs the constraint $X \succeq 0$ is satisfied on a subregion of the set $\mathbb{R}^{n\times n}$ of real $n \times n$ matrices. The same intuition holds for the constraint that X be orthogonal, have a fixed rank, or have a rank less than some fixed value: these all can be visualized as X lying in a subregion of $\mathbb{R}^{n\times p}$. These constraint-derived subregions will become the manifolds on which we will perform unconstrained optimization.

Algorithm 1: Gradient Descent

```
input  : A starting iterate $x^{(0)} \in \text{dom} f$
output: A minimizing sequence of iterates $x^{(k)}$
while stopping criteria not met do
    $\Delta x \leftarrow -\nabla f(x)$;
    Choose step size $t$ via line search;
    $x \leftarrow x + t\Delta x$;
```

The plan to generalize Algorithm 1 can be stated as follows in plain English:

1. Locally approximate the feasible set with an *inner product space*. That is, at each x in the feasible set $\mathcal{M}$ associate a *tangent space* $T_x\mathcal{M}$ with an inner product $\langle \cdot, \cdot \rangle_x$. These *tangent vectors* give us a way to compute the rate of change of scalar-valued functions of $\mathcal{M}$ in directions on $\mathcal{M}$. This turns $\mathcal{M}$ into a *Riemannian manifold.*

2. Compute the gradient of the function f_0 in the *ambient space* (e.g: $\mathbb{R}^n$ or $\mathbb{R}^{n\times p}$).

3. Project this gradient onto $T_x\mathcal{M}$ to get the rate of change of f_0 *along the manifold* in the direction of the gradient.

4. Map this projection back onto the manifold itself in a way that preserves this direction.

5. Repeat until convergence.

These steps are conceptual and need to be made much more precise if they are going to be used in the creation of implementable algorithms. The next three sections of this report are dedicated to this goal. In Section 2 we make the mathematical definition of a manifold precise. First we review necessary notions from topology and we discuss two ways to create new manifolds from existing ones: by *embedded submanifolds* and by *quotient manifolds*. Section 3 introduces the way in which we approximate manifolds with inner product spaces. This will also contain a way of calculating the gradients of scalar valued functions on manifolds. Section 4 rigorously defines retractions and presents the generalization of Algorithm 1 to manifolds.

With a generalized gradient descent developed we present some convergence results in Section 5. Section 6 contains examples of applying the present techniques to three problems and Section 7 concludes the report. In addition Appendix A contains the code listing to perform the experiments from Section 6.

We begin with important concepts from topology and geometry.

2 Topology, Geometry, & Manifolds

A smooth manifold $\mathcal{M}$ is a set that is locally Euclidean. That is, it can be mapped smoothly to open subsets of $\mathbb{R}^n$. This smoothness allows us to generalize concepts from calculus and apply them to the members of $\mathcal{M}$. Before we can define a manifold we need a precise concept of an open set that applies to both $\mathcal{M}$ and $\mathbb{R}^d$. This requires the notion of a *topology on a set.*

2.1 Topology

The goal of developing a topology on some set is to create abstractions of open sets in $\mathbb{R}^d$. While in $\mathbb{R}^d$ we often get open sets via norms, for abstract sets a topology gets constructed by fiat, subject to certain constraints on their behavior under unions and intersection.

Definition 2.1 (Topology, Topological Space)**.** A *topology* $\mathcal{T}$ on a set X is a collection of subsets of X such that :

1. $X, \emptyset \in \mathcal{T}$
2. $\cup_{\alpha \in I} T_\alpha \in \mathcal{T}$ where I is an arbitrary (possibly infinite) index set.
3. $\cap_{\alpha \in I} T_\alpha \in \mathcal{T}$ Where I is a finite index set.

The elements of $\mathcal{T}$ are called *open sets.* A *topological space* is a pair $(X, \mathcal{T})$ where X is a set and $\mathcal{T}$ is a topology on X.

These open sets are the direct analogues of open norm balls in $\mathbb{R}^d$ except they are more general; the shape of open sets is not restricted to being spherical and notions of *distance* are not in play. However the concept of points being "around" some other point is still important, and this leads to the concept of a neighborhood.

Definition 2.2 (Neighborhood)**.** If $(X, \mathcal{T})$ is a topological space and $x \in X$, then we define a *neighborhood* N of x to be a subset of X such that N includes an open set containing x.

Neighborhoods give a tool to describe local behavior. As can be inferred from the name they describe which points are around other points in the open sets of the topology. The existence of neighborhoods with specific local properties can be used to describe global properties of the topology. One such global property is the condition of being Hausdorff.

Definition 2.3 (Hausdorff Space)**.** A topological space $(X, \mathcal{T})$ is *Hausdorff* if, for any any two distinct points $x \neq y$ in X, there exist disjoint neighborhoods N_x, N_y of x and y respectively.

The Hausdorff condition can seem odd, but it guards against a potential pathology in constructing our minimization algorithms. In topological spaces that are not Hausdorff it is possible for a sequence $\{x_k\}$ of points in X to converge to two distinct limit points.[2] We say a sequence $\{x^{(k)}\}$ converges to $\tilde{x} \in X$ if for all neighborhoods N of $\tilde{x}$ there exists a positive integer L such that $x^{(\ell)} \in N$ for all $\ell \geq L$.

The last topological notion we need is that of a *basis*. Bases are similar in purpose to those in finite-dimensional vector spaces (FDVS). In FDVS a basis serves as a minimal set of vectors needed to recreate the entire space via linear combination. In a topological space a basis serves a a set that can recreate the whole topology via unions. This motivates the final topological definition.

Definition 2.4 (Basis, Second Countable)**.** A *basis* for a topology $\mathcal{T}$ on a set X is a collection of subsets B of X such that

1. For all $x \in X$, $x \in B_i$ for some $B_i \in \ B$.

2. For $x \in (B_i \cap B_j)$ where $B_i, B_j \in B$ there exists a B_k in B such that $x \in B_k$, and $B_k \subseteq B_i \cap B_j$.

   If $\mathcal{C}$ is a basis for $\mathcal{T}$ then $T = \{C : c = \cup_{B_i \in \mathcal{C}} B_i\}$, that is $\mathcal{T}$ is comprised of unions of elements of the basis. We say $(X, \mathcal{T})$ is *second countable* if $\mathcal{T}$ has a countable basis.

These notions are necessary for defining manifolds because they ensure the existence of a *Riemannian metric* which is crucial to how we define gradients in Section 3. These definitions are also key ingredients in how we define manifolds.

2.2 Manifolds

A manifold is a topological space where the topology arises from ways in which we map subsets of the topological space $\mathcal{M}$ to $\mathbb{R}^d$. To do this we need to specify a mapping for each subset of $\mathcal{M}$ to $\mathbb{R}^d$. This is done via a chart.

Definition 2.5 (Charts,Coordinates)**.** Let $\mathcal{M}$ be a set. A 1-1 mapping ϕ that takes a subset $\mathcal{U}$ of $\mathcal{M}$ and maps it to an open subset of $\mathbb{R}^d$, that is

$$\phi : \mathcal{U} \subseteq \mathcal{M} \to \mathcal{E} \subseteq \mathbb{R}^d, \mathcal{E} \text{ open}$$

is called a *d-dimensional chart* of $\mathcal{M}$. The coordinates of $\phi(x) \in \mathbb{R}^d$, $x \in \mathcal{U}$ are the coordinates of x in the chart ϕ. Often we'll write charts as a pair $(\mathcal{U}, \phi)$.

Clearly if we are going to have any success a single chart will not suffice (unless $\mathcal{U} = \mathcal{M}$). $\mathcal{M}$ must be covered in charts if a descent methods on $\mathcal{M}$ is going to work with any reasonable surety; we cannot at any point be unable to map our iterate to $\mathbb{R}^n$ or $\mathbb{R}$ because we've run out of charts. In addition we don't want this covering to introduce discontinuities in regions where the charts overlap. Such a collection of charts is called a C^∞ atlas.

Definition 2.6 (C^∞ Atlas)**.** A C^∞ *atlas* of $\mathcal{M}$ into $\mathbb{R}^d$ is a collection of charts $\{(\mathcal{U}_\alpha, \phi_\alpha)\}$ of $\mathcal{M}$ such that

1. $\cup_\alpha \mathcal{U}_\alpha = \mathcal{M}$

2. For all α, β such that $\mathcal{U}_\alpha \cap \mathcal{U}_\beta \neq \emptyset$ we have that both $\phi_\alpha(\mathcal{U}_\alpha \cap \mathcal{U}_\beta)$ and $\phi_\beta(\mathcal{U}_\alpha \cap \mathcal{U}_\beta)$ are open in $\mathbb{R}^d$ and $\phi_\beta \circ \phi_\alpha^{-1} : \mathbb{R}^d \to \mathbb{R}^d$ is C^∞.

For an atlas $\mathcal{A}$ the set $\mathcal{A}^+$ is the set of charts such that $\mathcal{A} \cup (\mathcal{U}, \phi)$ is an atlases. We call $\mathcal{A}^+$ the *maximal atlas* generated by $\mathcal{A}$.

[2]An example is given in [2] section 4.3.2. For the sake of space I won't review it here.

For a given set $\mathcal{M}$ it's possible for it to have many distinct C^∞ atlases. For a set to be a manifold we just need one such atlas where the topology induced by taking the $\mathcal{U}$'s to be the open sets is both Hausdorff and second-countable.

Definition 2.7 (Manifold)**.** A *d-dimensional manifold* is a pair $(\mathcal{M}, \mathcal{A}^+)$ where M is a set and $\mathcal{A}^+$ is a maximal C^∞ atlas of $\mathcal{M}$ onto $\mathbb{R}^d$ where the topology induced by $\mathcal{A}^+$ is both Hausdorff and second countable. The atlas is sometimes called a *differentiable structure* on $\mathcal{M}$.

In general there is no way to recognize or programmatically tell if a set $\mathcal{M}$ admits a differentiable structure without prescribing one. However, if we have a stock of known manifolds and a means of creating new manifolds from this stock (discussed in the subsection below) then the Riemannian approach is both highly general and widely applicable. For the purposes of this report we show that the familiar optimization realms — $\mathbb{R}^n$ and $\mathbb{R}^{n\times p}$ — are manifolds and demonstrate how we can construct differentiable structures on the feasible sets/feasible surfaces using this manifold structure.

The familiar domain of $\mathbb{R}^d$ is a manifold with any collection of open sets that creates a topology with the identity chart. That is we can turn $\mathbb{R}^d$ into a manifold with the chart $\{(\mathbb{R}^d, \mathrm{id}(x)\}$ where $\mathrm{id}(x) = x$. The natural extension of this is that *all* vector spaces are manifolds. If $\mathcal{E}$ is a d-dimensional Euclidean space with basis $\{e_j\}$ then the chart given by:

$$\begin{aligned} \phi : & \mathcal{E} \to \mathbb{R}^d \\ & x \mapsto (x_1, \ldots, x_d) \\ & \mathcal{U} \text{ arbitrary} \end{aligned}$$

where $x = \sum_i x_i e_i$ is a coordinate chart of $\mathcal{E}$ which turns it into a linear manifold.

A key immediate consequence of this is that certain sets of matrices are also linear manifolds. There are two which are of particular importance in this report:

$$\begin{aligned} \mathbb{R}^{n\times p} &:= \{ \text{ the set of all } n \times p \text{ real matrices}\} \\ \mathbb{R}^{n\times p}_* &:= \{ \text{ the set of all } n \times p \text{ real matrices with full column rank (} p \leq n)\} \end{aligned}$$

Since these are vector spaces over $\mathbb{R}$ they are both linear manifolds. In particular they are manifolds with a single chart $(\mathcal{U}, \phi)$ where $\mathcal{U}$ is $\mathbb{R}^{n\times p}$ or $\mathbb{R}^{n\times p}_*$ and the chart is given by the *vectorizing transform*:

$$\begin{aligned} \mathrm{vec} : \mathbb{R}^{n\times p} &\to \mathbb{R}^{np} \\ X &\mapsto [X_{1:}^T \ldots X_{p:}^T]^T \end{aligned}$$

where $X_{k:}$ is the k'th column X. Additionally we can turn these two sets into Euclidean space via the inner product:

$$\langle Z_1, Z_2 \rangle = \mathrm{vec}\,(Z_1)^T \,\mathrm{vec}\,(Z_2) = \mathrm{trace}\left(Z_1^T Z_2\right)$$

An issue with using this is that this transform destroys all matrix structure, of X. This is not necessarily desirable as matrix structure is part of what leads to the constraints in the first place. Thus we acknowledge that these sets are manifolds but we will prefer to work with their matrix as opposed to vectorized elements.

Since most constraints in optimization problems involve matrices or vectors, the feasible sets will be subsets of the *manifolds* $\mathbb{R}^n, \mathbb{R}^{n\times p}$, or $\mathbb{R}^{n\times p}_*$. We would like to avoid having to explicitly constructing a new atlass for each problem. The way we can accomplish this is to allow the feasible sets to inherit their manifold structure from their $\mathbb{R}^{n\times p}$ or $\mathbb{R}^d$. This is the topic of the next subsection.

2.3 New Manifolds From Old: Embedded Submanifolds & Quotient Manifolds

Assuming a fixed n, p with $p \leq n$ the relationship between $\mathbb{R}^{n\times p}$ and $\mathbb{R}^{n\times p}_*$ *as manifolds* is strongly linked to their relationship as sets. The relationship is that $\mathbb{R}^{n\times p}_*$ is a *submanifold* of $\mathbb{R}^{n\times p}$. What this means is, loosely, is that that one is a subset of the other and the chart from the latter induces a chart on the former. Since they are both manifolds in their own right (with the charts given above) they each can form the starting point for creating submanifolds.

This process of making new manifolds from old is key to the versatility of the Riemannian optimization framework. We discuss two ways of doing this: via *submanifolds* and via *quotient manifolds*. In the case of submanifolds we can recycle the charts (and Riemannian metric, as we will see) we already have. In the case of quotient manifolds, where we will create a new manifold via *equivalence classes*, more work needs to be done but as similar principle applies. Quotient manifolds will receive much less discussion for space reasons, as the definitions can be quite technical but the condition of being a quotient manifold has no impact on the convergence properties we discuss in Section 5. We start with submanifolds.

Suppose $(\mathcal{M}, \mathcal{A}^+)$ is a manifold and $\mathcal{X}$ is a subset of $\mathcal{M}$. While $\mathcal{X}$ as a set may admit several differentiable structures, as a subset of $\mathcal{M}$ it can admit only one. If $(\mathcal{M}, \mathcal{A}^+)$ and $(\mathcal{N}, \mathcal{C}^+)$ are manifolds such that $\mathcal{N} \subset \mathcal{M}$ then we consider $(\mathcal{N}, \mathcal{C}^+)$ to be a submanifold if the inclusion map $\iota : \mathcal{N} \to \mathcal{M}$, $\iota(x) = x$ has rank equal to the dimension of $\mathcal{N}$. This technical requirement can be interpreted as follows: the map ι that lets us consider $\mathcal{N}$ as a part of $\mathcal{M}$ does not expand or contract the dimension of $\mathcal{N}$. Another way to think of this is that we can go back and forth between thinking of elements of $\mathcal{N}$ as being in $\mathcal{N}$ and being in $\mathcal{M}$ freely.

A stronger condition is for the manifold topology of $\mathcal{N}$ under $\mathcal{C}^+$ to coincide with the subspace topology induced on $\mathcal{N}$ from viewing $\mathcal{M}$ as a *topological space* (taking the $\mathcal{U}$'s in the atlas for $\mathcal{M}$ to be the open sets.) When this occurs we say $\mathcal{N}$ is an *embedded submanifold* of $\mathcal{M}$. It can be shown (see [2]) that in this case it admits a unique differentiable structure with the charts on $\mathcal{N}$ given by $(\mathcal{N} \cap \mathcal{U}, \phi)$ where $(\mathcal{U}, \phi)$ is a chart of $\mathcal{M}$.

There are two key examples of embedded submanifolds that we will explore. The first is the n-sphere

$$S^{n-1} = \{x \in \mathbb{R}^n : ||x|| = 1\} \tag{2}$$

which is an embedded submanifold of $\mathbb{R}^n$. This has the natural chart of $\phi(x) = (x_1, \dots, x_n)$ and is useful for optimization problems with unit norm constraint. The other is the *compact* or *orthogonal* Stieffel manifold:

$$St(p, n) = \{X \in \mathbb{R}^{n\times p} : X^T X = I_p\}$$

In the case where $p = 1$ we have that $St(p, n) = S^{n-1}$ and when $n = p$ we recover O_n, the *orthogonal group.*

Quotient manifolds are significantly more abstract. They are constructed by lumping together all the points in a manifold $\mathcal{M}$ that are equivalent in some sense. These equivalent groups are called *equivalence classes.*

Definition 2.8 (Equivalence Relation, Equivalence Classes)**.** A binary relation $\sim$ on a set S is an *equivalence relation* if it satisfies the following properties for all $x, y, z \in S$

1. $x \sim x$ (reflexivity)

2. $x \sim y \implies y \sim x$ (symmetry)

3. $x \sim y, y \sim z \implies x \sim z$ (transitivity). The set

   $$[x] = \{y \in S : y \sim x\}$$

   is called the *equivalence class* of x under $\sim$.

For some set S the set of all equivalence classes of S under $\sim$ is called the *quotient* of S by $\sim$ and is written:

$$^{S}/_{\sim} := \{[x] : x \in S\}.$$

$^{S}/_{\sim}$ is the set of all equivalence classes of S with respect to $\sim$. Through some technical work these can also be given a differentiable structure which turns them into *quotient manifolds*.

Quotient manifolds are useful for representing large collections with a single member of the collection. This can have computational advantages in that it allows us to conserve computer memory. For example if we were try to represent the set of all lines through the origin in $\mathbb{R}^2$ we could use the equivalence relation $(x_1, y_1) \sim (x_2, y_2) \Leftrightarrow (x_1, y_1) = \alpha(x_2, y_2), \alpha \in \mathbb{R}$ to represent each line, which is an infinite collection of points, with a single vector.

This concept can be generalized from lines to p-dimensional subspaces or p-planes which yields the the *Grassmann manifold* Grass(p, n). In [2] the authors show that an equivalent way of writing the Grassmann manifold is

$$\begin{gathered}\text{Grass}(p, n) = {}^{\mathbb{R}_*^{n\times p}}/_{\sim} \\ X \sim Y \Leftrightarrow \text{span}(Y) = \text{span}(X)\end{gathered}$$

Quotient manifolds are useful in instances where we may not have isolated critical points. For example, if we care only about selecting the *optimal subspace* rather than the *optimal element* of that subspace, then optimizing on $Grass(p, n)$ is preferable to optimizing on $\mathbb{R}_*^{n\times p}$ as we will find an our optimal subspace without worrying how it's represented. A detailed discussion of how quotient manifolds can help with non-isolated critical points can be found in the second chapter of [2].

The above manifolds and others mentioned in the first four chapters of [2] are summarized in Table 1.

It is important to note that these two techniques are not an exhaustive list. There are other ways to create new manifolds — e.g *product manifolds* formed by taking the Cartesian product $\mathcal{M} \times \mathcal{N}$ of two manifolds $\mathcal{N}, \mathcal{M}$ — and the techniques can be combined. For example the authors in [6] use Newton's method on product manifolds with quotient structure for matrix completion.

We now turn to tangent spaces, confining our attention to embedded submanifolds.

3 Tangent Spaces: Vector Space Approximations to Manifolds

We mentioned earlier that the plan was to approximate manifolds locally by inner product spaces and we begin that process in earnest now. This approach is a natural generalization of gradient descent in $\mathbb{R}^n$ where the first order approximation to the cost surface in $\mathbb{R}^{n+1}$ given by $f(y) \approx f(x) + \nabla f(x)^T(y - x)$ approximates f at x with an affine space. Since the manifolds we're talking

Manifold Name	Notation	Set Description
Non-Compact Stieffel Manifold	$St_{nc}(p,n)$	$\{X \in \mathbb{R}^{n\times p} : \text{rank}(X) = p; p \leq n\}$
Orthogonal Stieffel Manifold	$St(p,n)$	$\{X \in R^{n\times p} : X^TX = I_p; p \leq n\}.$
The n-sphere	S^{n-1}	$\{x \in \mathbb{R}^n : \|\|x\|\| = 1\}$
The Oblique Manifold	$\mathcal{OB}$	$\{Y \in \mathbb{R}_*^{n\times p} : \text{diag}(YY^T) = I_n\}$[3]
Generalized Stieffel Manifold	$Stg(p,n)$	$\{X \in \mathbb{R}^{n\times p} : X^TBX = I_p, B = B^T, B \succ 0\}$
Symplectic Manifold	$Symp(n)$	$\{X \in \mathbb{R}^{2n\times 2n} : X^TJX = J, J = \begin{bmatrix} 0_n & I_n \\ -I_n & 0_n \end{bmatrix}\}$

Table 1: Table of Some Matrix Manifolds

about don't necessarily live inside R^n we will be able to approximate them with a true inner product space.

First we need a sense of direction on manifolds. The natural way to do this is via a parameterized curve $\gamma : \mathbb{R} \to \mathcal{M}$. It's tempting to define tangent vectors in a manner similar to ordinary calculus, that is:

$$\gamma'(t) = \lim_{h\to 0} \frac{\gamma(t+h) - \gamma(t)}{h}$$

but this fails because $\mathcal{M}$ itself *may not be a vector space* which means the operations in the limit expression may not be defined. To get around this we define a tangent vector as follows.

Definition 3.1 (Tangent Vector)**.** Let $\mathcal{F}_x(\mathcal{M})$ be the set of all smooth, real-valued functions defined on a neighborhood of x. A *tangent vector* ζ_x to $\mathcal{M}$ at x is a mapping from $\mathcal{F}_x(\mathcal{M}) \to \mathbb{R}$ such that there exists a curve $\gamma(t)$ on $\mathcal{M}$ where $\gamma(0) = x$ and $\zeta_x(f) = \dot{\gamma}(0)(f) = \dot{\gamma}(0)f = \frac{df(\gamma(t)}{dt}$ for all $f \in \mathcal{F}_x(\mathcal{M})$. We say that γ *realizes the tangent vector* ζ_x *and call* x *the foot of the tangent vector.*[4]

This definition is not particularly intuitive as it is hard to see how this connects with the usual idea of tangency. However, the definition as given *does* give us what we usually want out of first order approximations: a sense of the rate of change in a particular direction. The idea is that ζ_x is a function that takes as an input a real valued function $f \in \mathcal{F}_x(\mathcal{M})$ and outputs the rate of f along a specific curve $\gamma(t)$ in $\mathcal{M}$. That is, ζ_x gives the rate of change along γ for all smooth functions f on $\mathcal{M}$. Thus tangent vectors represent generalizations of directional derivatives.

The set of all tangent vectors at a point $x \in \mathcal{M}$ is in fact a vector space, called the *tangent space* and is notated $T_x\mathcal{M}$. We give it vector space status using the operations:

$$(a\dot{\gamma}_1 + b\dot{\gamma}_2)f = a(\dot{\gamma}_1(f)) + b(\dot{\gamma}_2(f))$$

for all $a, b \in \mathbb{R}$ and $\dot{\gamma}_1, \dot{\gamma}_2 \in T_x\mathcal{M}$.

As a final fact (proved in [2]) we note that if $\dim(\mathcal{M}) = d$ then $\dim(T_x\mathcal{M}) = d$ and we can give coordinate representations in $\mathbb{R}^n$ to $\dot{\gamma}$ in some chart $(\mathcal{U}, \phi), x \in \mathcal{U}$ by $\dot{\gamma} = (\dot{\gamma}(0)\phi_1, \ldots, \dot{\gamma}(0)\phi_d)$.

To get a set of approximations to the whole of $\mathcal{M}$ we "wrap" it in vector spaces and define the tangent bundle

$$T\mathcal{M} = \cup_{x\in\mathcal{M}} T_x\mathcal{M}$$

We note that $T\mathcal{M}$ is itself a manifold. To see this we use the fact that a tangent vector ξ_x is in $T_x\mathcal{M}$ for one and only one x. Using this relationship we can consider the tangent spaces of $\mathcal{M}$ as equivalence classes on $T\mathcal{M}$. From this relationship we turn TM into a manifold (by viewing $\mathcal{M}$ as a quotient manifold of $T\mathcal{M}$ and "lifting" it's manifold structure up to $T\mathcal{M}$).

[4]Requiring that $\gamma(0) = x$ is convention as the curve can always be reparameterized so that it's true.

Without proof we state formulations for the tangent spaces to S^{n-1} and $St(p,n)$ as they will be useful later.

$$T_x S^{n-1} = \{z \in \mathbb{R}^n : z^T x = 0\} \tag{3}$$

$$T_x St(p,n) = \{Z \in \mathbb{R}^{n\times p} : X^T Z + Z^T X = 0\} \tag{4}$$

We now turn this vector space into an inner product space.

3.1 Riemannian Metrics

Riemannian metrics allow us to give $T_x\mathcal{M}$ an inner product $\langle\cdot,\cdot\rangle_x$ for all $x \in \mathcal{M}$. From this we will derive an inner product norm on $T_x\mathcal{M}$ via

$$||\zeta_x|| := \sqrt{\langle\zeta_x, \zeta_x\rangle_x}$$

If for all $x \in \mathcal{M}$ we have an inner product $\langle\cdot,\cdot\rangle_x$ that varies smoothly over $\mathcal{M}$ we say that $\mathcal{M}$ is a *Riemannian Manifold* and that $\langle\cdot,\cdot\rangle_x$ is the *Riemannian metric* on $\mathcal{M}$. Three equivalent ways of writing the $\langle\cdot,\cdot\rangle_x$ are $g_x(\zeta_x, \zeta_x)$ and $g(\zeta_x, \zeta_x)$ with the last one being used when either x is clear from context or the Riemannian metric is identical in every tangent space. The topological definitions from Section 2 allows us to state the following useful lemma.

Lemma 3.1. *Any second countable Hausdorff manifold admits a Riemannian metric.*

While we have the existence guarantee from this lemma, there is no general formula for expressing the Riemannian metric for arbitrary manifolds. However, as in the case of charts, submanifolds can inherit metrics from their "parent" manifolds, so we attach Riemannian metrics to our stock of manifolds. Staying with the same first examples used in Section 2 we note that the Riemannian metric for $\mathbb{R}^d$ is:

$$\langle\zeta_x, \xi_x\rangle = g_x(\zeta_x, \xi_x) = g(\zeta_x, \xi_x) = \zeta_x^T \xi_x \tag{5}$$

and in $\mathbb{R}^{n\times p}$ and $\mathbb{R}_*^{n\times p}$ it is:

$$\langle A_x, B_x\rangle = g_x(A_x, B_x) = g(A_x, B_x) = \text{trace}\left(A_x^T B_x\right) \tag{6}$$

For embedded submanifolds we do no need to change the Riemannian metric, so for members of $T_x S^{n-1}$ we can simply use Equation (5) and similarly we can use Equation (6) for $T_x St(p,n)$. Quotient manifolds have an additional technical difficulty which is that members of the quotient manifold ${}^{\mathcal{M}}/_{\sim}$ may have infinitely many representations as members of $\mathcal{M}$. To circumvent this the metric is defined as being a unique element of $\mathcal{M}$ where tangent vectors do not change if you consider it as being part of $\mathcal{M}$ or ${}^{\mathcal{M}}/_{\sim}$. While we omit the details of this process remark briefly that the Riemannian metric on $Grass(n,p)$ is given by

$$g_{\text{span}(Y)}(Z_1, Z_2) = \text{trace}\left((Y^T Y)^{-1} Z_1^T Z_2\right)$$

With an inner product available we can describe orthogonal complements. These turn out to be very useful for defining gradients and performing line search on embedded submanifolds. Suppose $\mathcal{M}$ is an embedded submanifold of $\tilde{\mathcal{M}}$. We define the *normal space* to the $T_x\mathcal{M}$ as

$$(T_x\mathcal{M})^{\perp} = \{\zeta \in T_x\tilde{\mathcal{M}} : \bar{g}(\xi, \zeta) = 0\}$$

for all $\xi \in T_x\mathcal{M}$. Naturally we can then decompose vectors in $T_x\tilde{\mathcal{M}}$ via orthogonal projections. Let $\xi \in T_x\tilde{\mathcal{M}}$ then we can decompose it as

$$\xi = \underbrace{P_x\xi}_{\text{projection on to } T_x\mathcal{M}} + \underbrace{P_x^\perp\xi}_{\text{projection on to } (T_x\mathcal{M})^\perp}$$

This provides us with a useful computational tool: we can perform computations in $T_x\tilde{\mathcal{M}}$ (such as computing gradients) and then project onto $T_x\mathcal{M}$ via P_x to focus exclusively on the submanifold $\mathcal{M}$. Since we will use these tools in section 6 we give these quantities for S^{n-1} and $St(p,n)$ without proof.

- The embedded submanifold S^{n-1}:

$$T_xS^{n-1} = \{x : z^Tx = 0\} \tag{7}$$

$$(T_xS^{n-1})^\perp = \{\alpha x : \alpha \in \mathbb{R}\} \tag{8}$$

$$P_x\xi = (I - xx^T)\xi \tag{9}$$

$$P_x\xi = xx^T\xi \tag{10}$$

- The embedded submanifold $St(p,n)$:

$$T_xSt(p,n) = \{Z \in \mathbb{R}^{n\times p} : X^TZ + Z^TX = 0\} \tag{11}$$

$$(T_xSt(p,n))^\perp = \{XS : S \in \mathrm{Sym}(p)\} \tag{12}$$

$$\begin{aligned} P_x\xi &= (I - XX^T) - Xsym(X^T\xi) \\ &= (I - XX^T) + X^Tskew(X^T\xi) \end{aligned} \tag{13}$$

$$P_x^\perp\xi = Xsym(X^T\xi) \tag{14}$$

  where $skew(A) = \frac{1}{2}(A - A^T)$, $sym(A) = \frac{1}{2}(A + A^T)$ and $Sym(p)$ is the set of symmetric $p \times p$ matrices.

From now on we assume all manifolds are Riemannian.

If we are to define convergence rigorously we need a notion of distance, which we now provide.

3.2 Distances From Riemannian Metrics

Calling $\langle\cdot,\cdot\rangle$ a metric can be confusing as the inner product does not give a metric in the typical sense. To turn a Riemannian manifold into a true metric space we need something more. We define the length $L(\gamma)$ of a curve γ on $\mathcal{M}$ as

$$L(\gamma) := \int_a^b \sqrt{g(\dot\gamma(t), \dot\gamma(t))}dt$$

which then gives the metric:

$$\begin{gathered} \delta : \mathcal{M} \times \mathcal{M} \to \mathbb{R} \\ (x, y) \mapsto \inf_\Gamma L(\gamma) \end{gathered}$$

where Γ is the collection of all curves joining x and y. These definitions will be used to define convergence rates in Section 5.

Note that the metric is a function of g which means that defining $g(\cdot,\cdot) = \langle\cdot,\cdot\rangle_x$ gives us a metric without additional work. Thus in a sense calling $g(\cdot,\cdot)$ the metric is not too misleading as it is the key ingredient to defining genuine distances.

With a notion of distance we now turn to how to perform differential calculus on manifolds.

3.3 Differentiation on Manifolds

To define generalizations of gradients on manifolds we need a notion of differentiability on manifolds. To do this we need to define the *differential* of a mapping of maps between manifolds.

Definition 3.2 (Differential on a Manifold)**.** Let $\mathcal{M}, \mathcal{N}$ be manifolds and $F : \mathcal{M} \to \mathcal{N}$ smoothly. Suppose $\xi_x \in T_x\mathcal{M}$. It's possible to show that the mapping

$$\begin{aligned} DF(x)[\xi_x] :&\mathcal{F}_{F(x)}(\mathcal{N}) \to \mathbb{R} \\ &f \mapsto \xi(f \circ F) \end{aligned} \tag{15}$$

is a tangent vector to $\mathcal{N}$ at $F(x)$. We define the *differential of F at x* to be the map:

$$\begin{aligned} DF(x) :&T_x\mathcal{M} \to T_{F(x)}\mathcal{N} \\ &\xi \mapsto DF(x)[\xi] \end{aligned} \tag{16}$$

We have some useful applications of this theorem.

- If $\mathcal{N} = \mathbb{R}$ then $DF(x)[\xi_x] = \xi_x F$, that is applying the tangent vector to the real valued function F to get it's directional derivative.
- If $\mathcal{M}, \mathcal{N}$ are linear manifolds then

  $$DF(x)[\xi_x] = \lim_{t\to\infty} \frac{F(x + t\xi_x) - F(x)}{t}$$

  which is the traditional (Frechet) directional derivative.

Equation (15) lets us define gradients on manifolds.

Definition 3.3 (Gradient on Manifolds)**.** Define the gradient $\nabla f(x)$ of a real valued function f on a manifold $\mathcal{M}$ at a point $x \in \mathcal{M}$ to be the unique element of $T_x\mathcal{M}$ satisfying:

$$\langle \nabla f(x), \xi \rangle_x = Df(x)[\xi]$$

for all $\xi \in T_x\mathcal{M}$.

The following fact is useful for calculations in optimization problems and will be used heavily in Section 6.

Lemma 3.2. *If $\bar{f}$ is a real-valued cost function on a Riemannian manifold $\bar{\mathcal{M}}$ and f is the restriction of $\bar{f}$ to an embedded submanifold $\mathcal{M}$ of $\bar{\mathcal{M}}$ then the gradient of f is the orthogonal projection of the gradient of $\bar{f}$ onto $T_x\mathcal{M}$, that is:*

$$\nabla f(x) = P_x \nabla \bar{f}(x)$$

Next we turn generalizing Algorithm 1.

4 Retractions & Line Search on Manifolds

We are almost in a position to describe line-search algorithms on manifolds. The only difficulty is making sure that our iterates remain on $\mathcal{M}$, because Theorem 3.2 gives us gradients in the *tangent space* which is not necessarily in $\mathcal{M}$. To circumvent this difficulty we take steps determined by the gradient projected onto the tangent space and then return back to the manifold via special projection-type mappings called *retractions.*

Intuitively, a retraction is a mapping from the tangent space $T_x\mathcal{M}$ to $\mathcal{M}$ that preserves directionality at x. The idea is to take the gradient of our cost function f (by projection onto $T_x\mathcal{M}$) take a step in this direction, then retract back to get a new point on $\mathcal{M}$, and repeat.

Definition 4.1 (Retraction)**.** A *retraction* R is a smooth mapping from the tangent bundle $T\mathcal{M}$ to $\mathcal{M}$ with the following properties. If R_x is the restriction of R to $T_x\mathcal{M}$ and 0_x is the zero element of $T_x\mathcal{M}$ then

1. $R_x(0_x) = x$

2. With the identification $T_{0_x}T_X\mathcal{M} = T_x\mathcal{M}$[5] $R_x(0_x) = \mathrm{id}_{T_x\mathcal{M}}$.

From now on we assume that the domain of R is the entire tangent bundle. The second condition above can be restated in the following more intuitive way: for all $\xi \in T_x\mathcal{M}$ the curve γ_ξ mapping t to $R_x(t\xi)$ satisfies $\dot{\gamma}(0) = \xi$. This means that moving along γ_ξ is moving in the direction ξ while staying on the manifold. The last task for our approach now are to choose our descent direction and step size. We choose our step direction so that it's *gradient related* and the step size is chosen via an adapted version of the Armijo-Wolfe line search.

Definition 4.2 (Gradient-Related Sequence)**.** Given a cost function f on a Riemannian manifold M a sequence $\{\eta_k\}, \eta_k \in T_{x_k}\mathcal{M}$ is *gradient-related* if for any subsequence $\{x_k\}_{k\in\mathcal{K}}$ that converges to a critical point of f, the corresponding $\{\eta_k\}_{k\in\mathcal{K}}$ is bounded and satisfies:

$$\limsup_{k\to\infty, k\in\mathcal{K}} \langle \nabla f(x_k), \eta_k \rangle < 0$$

Clearly $\eta_k = -\nabla f(x_k)$ is gradient related and we will use that to be our step direction in the experiments in Section 6 and in the convergence analysis in Section 5.

Definition 4.3 (Armijo Point)**.** Given a cost function for a Riemmanian manifold with retraction R, a point $x \in \mathcal{M}$, a tangent vector $\xi_x \in T_x\mathcal{M}$, and scalars $\bar{\alpha} > 0, \beta, \sigma \in (0,1)$ the *Armijo point* is $\eta^A = t^A\eta = \beta^m\bar{\alpha}\eta$ where m is the smallest non-negative integer such that

$$f(x) - f(R_x(\beta^m\bar{\alpha}\eta)) \geq -\sigma \langle \nabla f(x), \beta^m\bar{\alpha}\eta \rangle_x \tag{17}$$

With these definitions in hand we can finally write the pseudocode for gradient descent on manifolds. This is given in Algorithm 2.

A convenient reformulation of Equation (17) is

$$f(R_x(\beta^m\bar{\alpha}\eta) \leq f(x) + \sigma \langle \nabla f(x), \beta^m\bar{\alpha}\eta \rangle_x \tag{17'}$$

which allows us to write Algorithm 2 in a more a way that makes it clear what needs to be provided for practical implementation. This is given in Algorithm 3.The two algorithms are equivalent and will be referenced interchangeably.

[5]This is a technical way of writing the requirement that the tangent space at 0_x to the tangent space is just the tangent space at x

Algorithm 2: Accelerated Line Search On Manifolds

```
input  : A Riemannian manifold $\mathcal{M}$; a continuously differentiable cost function $f$ on $\mathcal{M}$,
         scalars $\alpha\, 0, c, \beta, \sigma \in (0,1).$, an initial iterate $x^{(0)}$, a retraction $R$ on $T\mathcal{M}$.
output: A sequence of $x^{(k)}$ on the manifold $\mathcal{M}$.
while stopping criteria not met do
    Pick $\eta_k \in T_{x^{(k)}}\mathcal{M}$ such that the sequence $\{\eta_i\}_{i=0,1\ldots,}$ is gradient-related.;
    Select $x_{k+1}$ such that
        $$f(x_k) - f(x_{k+1}) \geq c(f(x_k) - f(R_{x_k}(t_k^A \eta_k))$$
    where $t^A$ is the Armijo step size.;
```

Algorithm 3: Reformulation of Accelerated Line Search On Manifolds

```
input  : A Riemannian manifold $\mathcal{M}$; a continuously differentiable cost function $f$ on $\mathcal{M}$,
         scalars $\alpha > 0, c, \beta, \sigma \in (0,1).$, an initial iterate $x^{(0)}$, a retraction $R$ on $T\mathcal{M}$.
output: A sequence of $x^{(k)}$ on the manifold $\mathcal{M}$.
while stopping criteria not met do
    $\eta_k \leftarrow -\nabla f(x)$ where $\nabla f(x)$ is the gradient of $f$ projected onto the tangent space of $\mathcal{M}$ ;
    $s \leftarrow \beta$;
    while $f(R_x(\alpha s \eta_k)) > f(x^{(k)}) - \sigma \langle \nabla f(x), s\alpha\eta_k \rangle_x$ do
        $s \leftarrow s\beta$;
    $x^{(k+1)} \leftarrow R_x(\alpha s \eta_k)$;
```

Having presented the generalization of Algorithm 1 on manifolds we turn to showing convergence results.

5 Convergence analysis

First we recall the definition of convergence. We will say that an infinite sequence $\{x_k\}$ of points in $\mathcal{M}$ is *convergent* if there exists a chart $(\mathcal{U}, \phi)$ of $\mathcal{M}$, a point $x^* \in \mathcal{M}$ and a $K > 0$ such that for all $k \geq K$ we have $x_k \in \mathcal{U}$ and the sequence $\{\phi(x_k)\}$ in $\mathbb{R}^d$ converges to $\phi(x^*)$. The point $\phi^{-1}(\lim_{k\to\infty} \phi(x_k))$ is called the *limit* of $\{x_k\}$. Since we assume $\mathcal{M}$ is Hausorff the limits are unique.

5.1 Critical Point Convergence & Stability

The first crucial point in our analysis is the fact that our line search algorithm is guaranteed to cluster around critical points of the cost function.

Theorem 5.1 (Critical Point Limit Guarantee)**.** *Let $\{x_k\}$ be an infinite sequence of iterates generated by Algorithm 2 . Then every limit point of $\{x_k\}$ is a critical point of the cost function f.*

The proof of Theorem 5.1 is too long to reproduce here, but we can summarize it briefly. The general approach of the proof is to show that if the sequence of iterates does not converge to a critical point then at some in creating the sequence we did not choose a step according to the Armijo condition. We can remedy this by normalizing that step η_k, but then taking limits we find that the sequence was not gradient related, which is a contradiction.

Theorem 5.1 is less than what one would want from an optimization guarantee as we do not rule out saddle points or maxima. However, this kind of information is difficult to obtain using just first-order approximations in non-convex settings. What we *can guarantee* is that the desirable critical points — that is, local minimizers — are *stable* under repeated applications of *descent mappings.*

Definition 5.1 (Descent Mapping)**.** A descent mapping F for a cost function f if

$$f(F(x)) \leq f(x)$$

for all $x \in \mathcal{M}$.

Descent mappings are generalizations of the concept of descent directions in $\mathbb{R}^n$. Notions of stability refer to the behavior of sequences generated by repeatedly applying descent mappings.

Definition 5.2 (Stability)**.** Let $F : \mathcal{M} \to \mathcal{M}$ be a descent mapping. A point $\tilde{x} \in \mathcal{M}$ is a *fixed point* of F if $F(\tilde{x}) = \tilde{x}$. We define $F^{(n)}$ to be the application of F n times. We say a fixed point $\tilde{x}$ of F is *stable* if for every neighborhood $\mathcal{O}$ of $\tilde{x}$ there is a neighborhood $\mathcal{V}$ of $\tilde{x}$ such that for all $x \in \mathcal{V}$ and all positive integers n it holds that $F^{(n)}(x) \in \mathcal{U}$. We call $\tilde{x}$ *asymptotically stable* if it is stable and $\lim_{n\to\infty} F^{(n)}(x) = \tilde{x}$ for all x sufficiently close to $\tilde{x}$. A point is *unstable* if it is not stable, meaning that for all neighborhoods $\mathcal{G}$ of $\tilde{x}$ there exists a point $x \in \mathcal{G}$ such that $F^{(n)} \notin \mathcal{G}$ for some n.

The intuition behind these definitions is fairly clear. A stable point is one where you cannot easily leave the neighborhood $\tilde{x}$ because iterating F in all neighborhoods will tend back towards $\tilde{x}$. Unstable points have a "way out" via $\mathcal{G}$.

The following theorem provides a useful stability result on manifolds.

Theorem 5.2 (Only Local Minimizers are Stable)**.** *Suppose $F : \mathcal{M} \to \mathcal{M}$ is a descent mapping for a smooth cost function f on $\mathcal{M}$ and assume that for all $x \in \mathcal{M}$ the accumulation points of the sequence generated by repeatedly applying F (that is $\{F^{(k)}\}_{k=1}^{\infty}(x)$) are critical points of f. Suppose x^* is a fixed point of F and that x^* is not a local minimizer of f. If there is a compact neighborhood $\mathcal{U}$ of x^* where for every critical point y of $f \in \mathcal{U}$, $f(y) = f(x^*)$ then x^* is an unstable point for F.*

The intuition here is that if x^* is not a local minimizer and there is a compact neighborhood with critical points that have the same value under f as x^* then these provide a way for the sequence $\{F^{(k)}(x)\}$ to move away from x^* and cause it to be unstable. The proof is simple and done via contradiction.

Proof. By the assumption that x^* is not a local minimizer, we must have a point y where $f(y) < f(x)$ where y is in some neighborhood $\mathcal{V}$ of x^*. We construct $z_k = F^{(k)}(y)$ and suppose that z_k is in the compact neighborhood $\mathcal{U}$ for all k (if it is not then x^* is immediately unstable). The compactness assumption means that z_k has an accumulation point $\tilde{z}$ in $\mathcal{U}$. By the stipulations of the theorem we have that $\tilde{z}$ is a critical point of f and therefore $f(\tilde{z}) = f(x^*)$. However, we stipulated that F is a descent mapping. This means that $f(\tilde{z}) \leq f(y)$ but we also have that $f(y) < f(x^*)$. This contradicts the fact that that $f(z) = f(x^*)$ which means that x^* does not satisfy the stability definition. □

This theorem has as an immediate consequence the *capture theorem* which says that isolated local minimizers of smooth cost functions are asymptotically stable.

Theorem 5.3 (Capture Theorem)**.** *If $F : \mathcal{M} \to \mathcal{M}$ is a descent mapping for a smooth cost function f and the limit points of $\{F^{(k)}\}_{k=1}^{\infty}$ are all critical points of f. Let x^* be a local minimizer and an* *isolated critical point of f. Assuming that $\delta(F(x), x) \to 0$ as $x \to x^*$ then x^* is an asymptotically stable of F.*

We omit the proof of Theorem 5.3 but note that it shows that Algorithm 2 will be stable around isolated local minimizers. We turn now to convergence rates.

5.2 The Speed of Convergence Of Manifold Line-search Algorithms

Here we review two notions of convergence rates as they are defined on manifolds, and comment about their relevance to iterative algorithms in the style of Algorithms 2 and 3. We begin with linear convergence.

Definition 5.3 (Linear Convergence)**.** Let $\mathcal{M}$ be a Riemannian manifold and let δ be the associated Riemannian metric. A sequence $\{x^{(k)}\}$ is said to converge *linearly* to a point $x^* \in \mathcal{M}$ if there exists a constant c and an integer $K \geq 0$ such that, for all $k \geq K$, it holds that

$$\delta(x^{(k+1)}, x^*) \leq c\delta(x^{(k)}, x^*) \tag{18}$$

In describing an iterative algorithm on $\mathcal{M}$, we say that it converges *locally linearly to x^** if there exists a neighborhood $\mathcal{U}$ o x^* and a constanct $c \in (0, 1)$ such that for every initial $x^{(0)} \in \mathcal{U}$ the sequence of iterates generated by the algorithm satisfies Equation (18).

The definition of linear convergence required a Riemannian metric, which means that the definition above is not as general as it could be. However, it has a useful related lemma above convergence in the tangent bundle.

Lemma 5.4 (Retraction Linear Convergence Lemma)**.** *A convergence sequence $\{x^{(k)}\}$ on a Riemannain manifold $\mathcal{M}$ converges linearly to x^* with constant c if and only if*

$$||R_{x^*}^{-1}(x^{(k+1)}) - R_{x^*}^{-1}(x^*)|| \leq c||R_{x^*}^{-1}(x^{(k)}) - R_{x^*}^{-1}(x^*)||$$

where R is any *retraction on $\mathcal{M}$.*

The point of this lemma is that we can use convergence in the tangent bundle (the domain of R) as a proxy for convergence on the manifold in terms of the Riemannian metric. This means that algorithms in the style of Algorithm 2 can use retractions without worrying about changing intrinsic convergence results.

We can define superlinear convergence without referencing the Riemannian metric at all; all we need is the set of charts.

Definition 5.4 (Superlinear Convergence)**.** Suppose $\mathcal{M}$ is a manifold (*not* necessarily Riemannian) and $\{x^{(k)}\}$ is a sequence on $\mathcal{M}$ converging to some point x^*. Let $(\mathcal{U}, \phi)$ be a chart of $\mathcal{M}$ with $x \in \mathcal{U}$. We say $\{x^{(k)}\}$ converges *superlinearly* to x^* if

$$\lim_{k \to \infty} \frac{||\phi(x^{(k+1)}) - \phi(x^*)||}{||\phi(x^{(k)}) - \phi(x^*)} = 0 \tag{19}$$

If we have constants $p > 0, c \geq 0$ and $K \geq 0$ such that for all $k \geq K$ we have that

$$||\phi(x^{(k+1)}) - \phi(x^*)|| \leq c||\phi(x^{(k)}) - \phi(x^*)||^p \tag{20}$$

then we say $\{x^{(k)}\}$ converges to x^* with order at least p.[6]. An iterative algorithm on $\mathcal{M}$ is said

[6]Note that the norm here is the norm on $\mathbb{R}^d$

to converge *locally to x^* with order at least p* if there's a chart $(\mathcal{U}, \psi)$ at x^* and a constant $c > 0$ such that for every initial iterate $x^{(0)} \in \mathcal{U}$ the sequence generated by the algorithm satisfies the inequality in (20). As can be anticipated we say the convergence is quadratic if $p = 2$ and cubic if $p = 3$.

While these definitions are intuitive generalizations of the usual notions, we don't necessarily observe them frequently in practical applications of iterative algorithms on manifolds. The reason for this is that the way in which we choose the step an iterative algorithm is not differentiable. This can be made more precise via the following theorem. We omit the proof.

Theorem 5.5 (Convergence Rate Guarantees)**.** *Suppose $F : \mathcal{M} \to \mathcal{M}$ is the mapping that encodes how we choose our descent step in an iterative algorithm. Suppose further that the domain of F contains a neighborhood of x^* which we take to be a fixed point of F. Then the following hold:*

- *If the derivative of F at x^* is equal to 0 and F is C^1 then an iterative algorithm using F as the way it chooses its next iterate converges locally superlinearly to x^*.*
- *If the the derivative of F is 0 at x^* and F is C^2, then the algorithm converges locally quadratically to x^*.*

Differentiability here is on the manifold $\mathcal{M}$ *in the sense defined in Section 3.3.*

Smooth mappings, however, are rare in iterative algorithms, and so we can't expect Algorithms 2 and 3 have superlinear convergence.

The closing theorem of our convergence analysis gives gives a rate of convergence for line search on algorithm on manifolds. Because it's proof requires the theory of the eigenvalues of a Hessian on a Riemannian manifold, we do not give the proof here.

Theorem 5.6 (Convergence Rate of Algorithm 2)**.** *Let $\{x^{(k)}\}$ be an infinite sequence of iterates generated by Algorithm 2 where η_k is chosen to be $-\nabla f(x_k)$ and $\{x^{(k)}\}$ converges to some point x^*. Let $\lambda_{H,\min}$ and $\lambda_{H,\max}$ be the smallest and largest eigenvalues of the Hessian of f at x^*. Assume that $\lambda_{H,\min} > 0$ and we are given an r in the interval $(m, 1)$ where $m = 1 - \min(2\sigma\bar{\alpha}\lambda_{H,\min}, 4\sigma(1-\sigma)\beta\frac{\lambda_{H,\min}}{\lambda_{H,\max}})$ where $\sigma, \beta, \bar{\alpha}$ are the parameters for Algorithm 2. Then there exists an integer $K \geq 0$ such that*

$$f(x^{(k+1)}) - f(x^*) \leq (r + (1-r)(1-c))(f(x^{(k)}) - f(x^*)) \tag{21}$$

This gives an analogous result to the convergence analysis for gradient descent (see [4]) and means that

$$f(x^{(k)}) - f(x^*) \leq (r + (1-r)(1-c))^k (f(x^{(0)}) - f(x^*)) \tag{22}$$

or

$$f(x^{(k)}) - f(x^*) \leq r^k (f(x^{(0)}) - f(x^*)) \tag{23}$$

when we take $c = 1$. To show how to adapt these algorithms to real world problems we now provide some examples and computational experiments. In those experiments we take the c from Algorithm 2 to be 1 for simplicity.

6 Examples and Experiments

We present three examples of using the above theory to solve constrained optimization problems.

6.1 The Rayleigh Quotient on the Sphere

Consider the following optimization problem

$$\begin{aligned} \underset{x\in\mathbb{R}^n}{\text{minimize}} \quad & x^TAx \\ \text{subject to} \quad & ||x|| = 1 \end{aligned} \tag{A}$$

where A is a real symmetric matrix, but not necessarily positive (semi)definite. The authors in [2] refer to this problem as minimizing the *Rayleigh quotient on the sphere.* The authors show that the minimizer of Equation (A) is λ_1, the smallest eigenvalue of A. Our constraint set $\{x \in \mathbb{R}^n : ||x|| = 1\}$ is exactly the unit sphere S^{n-1} which as we showed above is an embedded submanifold of $\mathbb{R}^n$, and so we will approach solving (A) via Algorithm 3. In order to do this we need two components:

1. A formula for the projected gradient of x^TAx along the tangent space to S^{n-1} and
2. A retraction $R_x(\cdot)$.

To compute the first item we remark that on $\mathbb{R}^n$ the gradient of $f(x) = x^TAx$ is

$$2Ax$$

The using the formula in (9) we compute the projected gradient along the tangent space to be:

$$P_x(2Ax) = 2(Ax - xx^TAx) \tag{24}$$

So our step direction at each iterate $x^{(k)}$ will be

$$\eta_k \leftarrow -2(Ax^{(k)} - x^{(k)}(x^{(k)})^TAx^{(k)})$$

For the retraction we have two options. The first, which I will call the *basic* retraction as it is just normalizing the vector:

$$R^b_x(\xi) = \frac{x+\xi}{||x+\xi||}$$

and the retraction that can be derived from the *exponential map*[7]:

$$R^e_x(\xi) = x\cos(||\xi||) + \frac{\xi}{||\xi||}\sin(||\xi||)$$

Using R^e_x is the same as retracting to anywhere on a great circle on S^{n-1} as opposed to simply normalizing the vector. The results of running Algorithm 3 on this problem are show in Figures 1 to 3.

In Figures 1 and 2 we show the performance of the method for random $n \times n$ symmetric matrices and for the diagonal matrix with diagonal entries $1, \ldots, n$, respectively, for various sizes of n. In these simulations we set $\beta = 0.5, \alpha = 1, \sigma = 0.5$ and ran until we had a duality gap less than 10^{-8} or we had reached 1000 iterations. In both settings the methods converged rapidly for small matrices but performance degraded significantly as matrix dimensions grew larger. In the case of random symmetric matrices after around a size of 20×20 the performance degraded and the rate of convergence slowed to roughly linear convergence. For the diagonal matrices we observed a slowdown starting at 500×500. The higher threshold in this case might be due to the sparsity of $diag(1, \ldots, n)$.

[7]Defined in chapter 5 of [2]

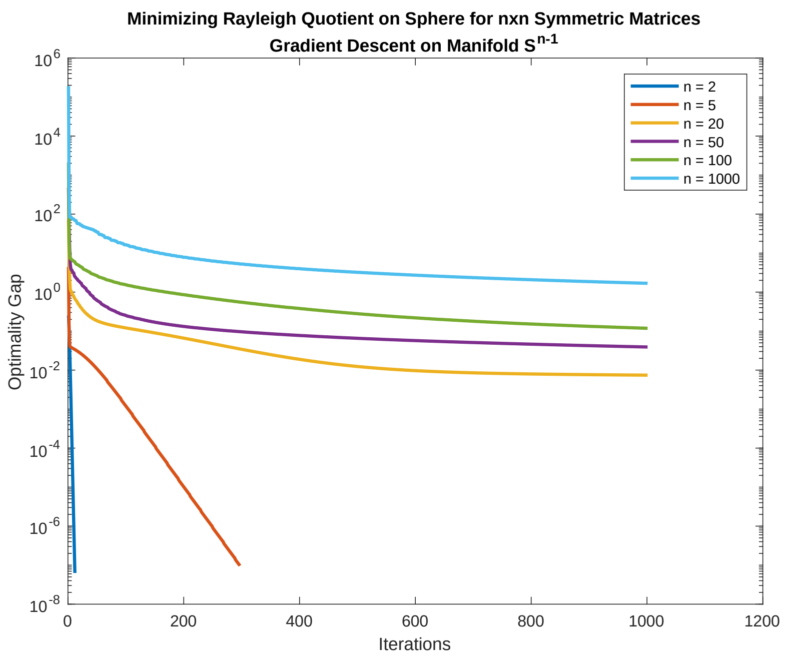

Figure 1: Running Algorithm 3 on Equation (A) for random $n \times n$ symmetric matrices.

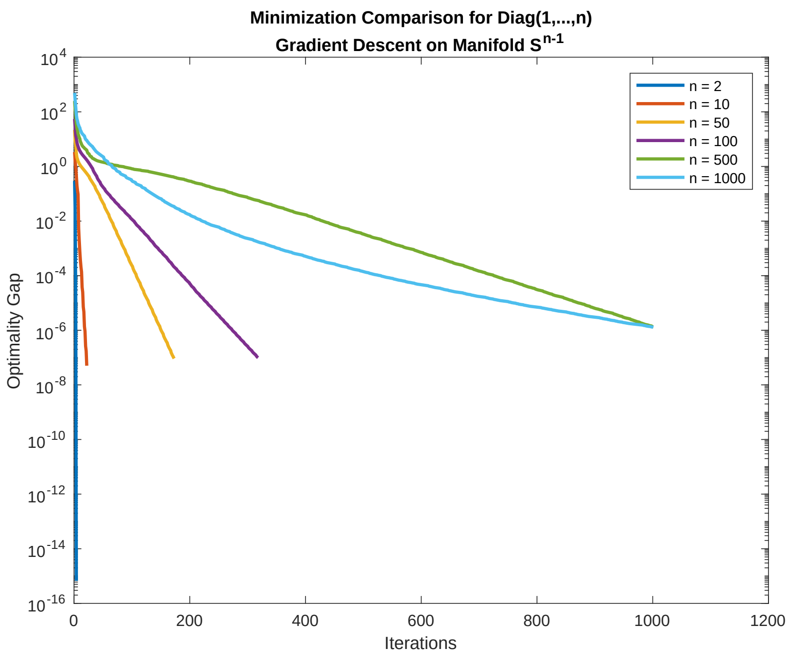

Figure 2: Running Algorithm 3 on Equation (A) for the matrix $\mathrm{diag}(1, \ldots, n)$ for various values of n.

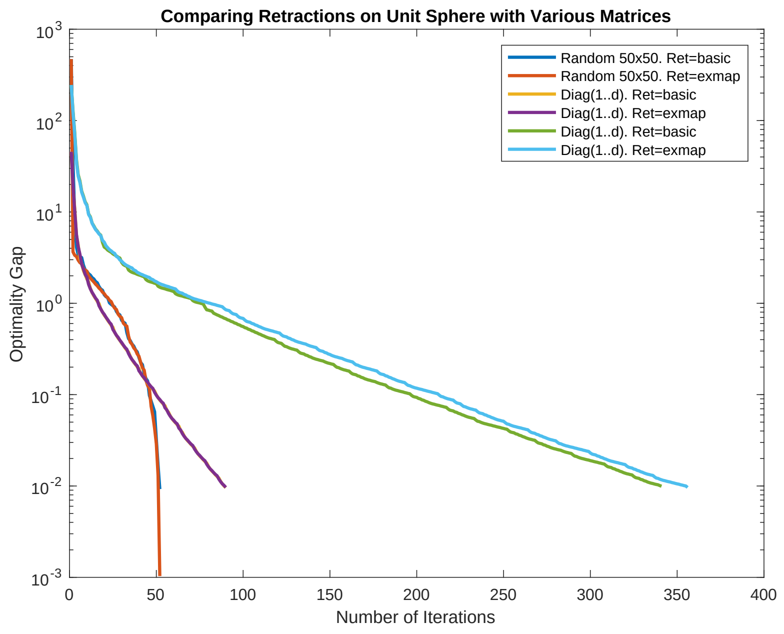

Figure 3: Comparing the two retractions on the sphere.

Figure 3 Compares the performance of the same algorithm using the retractions R^e and R^b in various settings. In all cases the retractions behaved similarly and had almost identical optimality gaps at each iteration. Here we changed the set up running until we had reached an optimality of less than 10^{-3} or 1000 iterations.

In all cases to generate the initial iterate $x^{(0)}$ we generated a random vector in $\mathbb{R}^n$ and divided it by its norm. The code listings for these figures and the algorithms are in Appendix A.

The next example is a calculation of the necessary methods for a different problem on S^{n-1}.

6.2 Directional Least Squares

As the previous problem is discussed in [2] we present a computation without experiments for a different problem, which we call the *directional least squares problem.* Suppose we have a matrix of observed data A and outcomes b. Then the directional least squares problem can be written as:

$$\begin{aligned} \underset{x\in\mathbb{R}^n}{\text{minimize}} \quad & \frac{1}{2}||Ax-b||^2 \\ \text{subject to} \quad & ||x|| = 1 \end{aligned} \tag{B}$$

Solving this is equivalent to finding the *direction* in $\mathbb{R}^n$ of the minimizer x^* without regard for the values of the actual components. Since we are working with the same underlying manifold S^{n-1} we have the same retractions available and only need to compute the gradient. The gradient of $\frac{1}{2}||Ax-b||^2$ on $\mathbb{R}^n$ is given by

$$A^T(Ax-b)$$

Using the formula given in (9) this then gives us a projected gradient

$$P_x(A^T(Ax-b)) = (A^T(Ax-b)) - xx^T(A^T(Ax-b))$$

We can then simply set

$$\eta_k \leftarrow -\left[(A^T(Ax-b)) - xx^T(A^T(Ax-b))\right]$$

and minimize with Algorithm 2.

While this is in a certain sense a toy problem, it demonstrates the versatility of this framework. Though we have a non-convex constraint, we are able to perform a simple calculation and use already-established machinery (in terms of the projection in (9) and the retractions above) to quickly adapt Algorithm 3 to a new problem. By equipping the the constraint set with this additional machinery new problems are easily adapted by focusing on the objective function f_0 alone.

We did not perform any experiments with this problem as directional data sets are not widely available and with typical regression data sets the direction is *less* important than the actual vector. As a result any solution to (B) would likely perform worse in a statistical learning setting.

For the final example we turn to the orthogonal Stieffel manifold $St(p,n)$.

6.3 The Procrustes Problem

We consider a version of the Procrustes problem:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & ||XW - Z||_F \\ \text{subject to} \quad & X^TX = I \end{aligned} \tag{C}$$

where all matrices are assumed to be $n \times n$. As before we need a projected gradient as well as a retraction. Going in reverse order we remark, without proof, that a suitable retraction for $St(p,n)$ is

$$R_X(\xi) = qf(X+\xi)$$

where $qf(\cdot)$ represents the q factor of the QR decomposition of $X+\xi$.

The computation of the gradient is more involved. Elementary differentiation shows that the derivative of $||XW-Z||_F$ is :

$$\frac{1}{||XW-Z||_F}(XW-Z)W^T$$

Applying the projection from (13) and letting $M = (XW-Z)$ we get the projected tangent space gradient as

$$\begin{aligned} P_X(\frac{M}{||M||_F}W^T) &= \frac{M}{||M||_F}W^T - X * sym(X^T\frac{M}{||M||_F}W^T) \\ &= \frac{M}{||M||_F}W^T - X\left[\frac{1}{2}(\frac{X^TMW^T}{||M||_F} + \frac{(X^TMW^T)^T}{||M||_F}\right] \\ &= \frac{M}{||M||_F}W^T - \frac{X}{2||M||_F}\left[X^TMW^T + WMX\right] \end{aligned}$$

To experiment with this we used a face from the Olivetti faces dataset ([1]). The dataset is stored as a set of 64×64 matrices with entries encoding grayscale values. For a single face $F \in \mathbb{R}^{64\times 64}$ we generated a random rotation (orthogonal) matrix R by performing the QR factorization of a random 64×64 matrix. We then set W in (C) to be RF and Z to be F. Thus solving (C) is equivalent to finding the inverse rotation to R. If X^* is a solution to (C) then we should have that $XW = XRF = F$. The results of this approach are shown in Figure 4.

The results are fairly encouraging. We have visibly recovered the major features of the face,

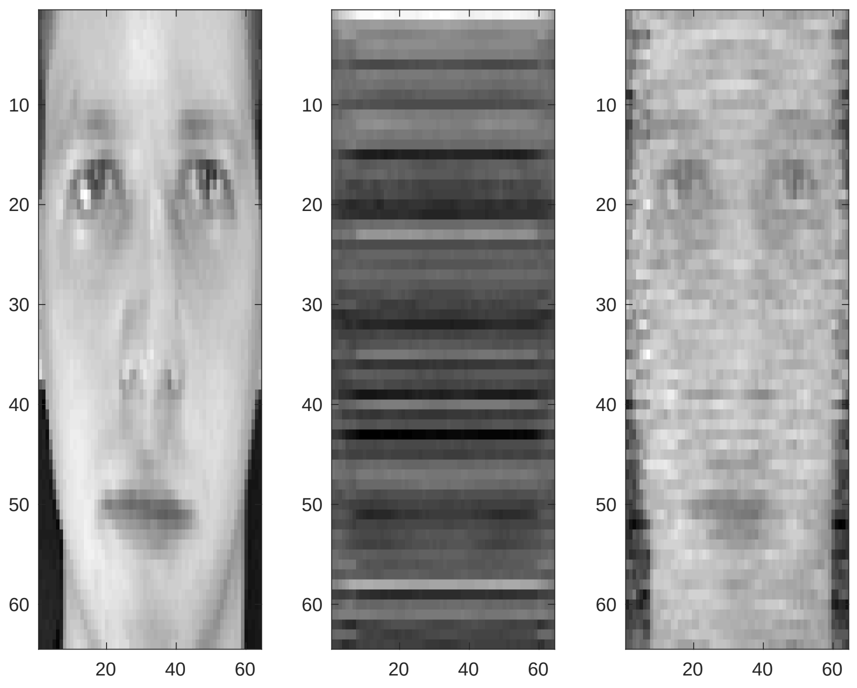

Figure 4: Running Algorithm 2 on Equation (C). The left is the original face, the center after the first rotation, and the right is the output of the algorithm applied to the center image.

but the reconstruction is not perfect. The discrepancy between the original image and the image on the far right can be attributed to the fact that because we are using an iterative method we may get very close to the true solution without getting there explicitly. This small discrepancy can result in the distortion we see.

What is interesting is that an iterative, first order descent method has found something that looks at all like a solution. The particular form of (C) is that of an *orthogonal Procustes problem* which is known to have an analytic solution ([9])

$$X^* = UV^T$$

where U and V come from the singular value decomposition of ZW^T. Thus in solving (C) we've found the inverse of R and also performed an approximation to the singular value decomposition of ZW^T. As in the first problem the code used to generate this figures is in Appendix A.

7 Conclusion & Related Work

We've seen how first order methods on matrix manifolds can be used to solve constrained opitimization problems. Along the way we reviewed necessary notions from topology and Riemannian geometry, defined a generalized version of gradient descent on Riemannian manifolds, and provided some convergence results.

This is by no means the end of the theory. In [2], chapters 5 & 6 develop a generalization of Newton's method for Riemannin (matrix) manifolds. The subsequent two chapters develop trust region methods and a collection of super-linear[8] on algorithms based on a concept called *vector transport*.

This mature theory has enabled a growing field of applied and theoretical research. There are at least two actively maintained software packages for performing manifold optimization ([3],[10]). The package [3] was used by the authors in [8] to address the problem of synchronization over the special Euclidean group, a problem which has applications to robotics and signal processing for imaging. In the field of statistical estimation the authors in [5] use Riemannian optimization to fit Gaussian mixture models.

There also exists a theoretical effort to generalize classic optimization algorithms to manifolds. For example the authors in [7] generalize the ADMM algorithm to Manifolds. In addition the authors of [2] have numerous theoretical and applied papers published in the field of optimization methods on matrix manifolds.

As a final point on applications we note that the authors of [2] claim that the algorithms for solving eigenvalue problems that use the techniques of optimization on matrix manifolds are competitive with state of the art methods. The only experiment discussed in this report comes from solving Equation (A). In those experiments we observed rapid convergence for smaller matrices, but convergence rapidly slowed as the dimension of the matrix increased. It's also worth noting that we were focusing on a single eigenvalue of a real symmetric matrix, which may be far from the most challenging case. It is possible, however, that applying more sophisticated manifold methods or extending the theory to *complex manifolds* (where the charts map to $\mathbb{C}^n$ instead of $\mathbb{R}^n$) would create competitive algorithms, but that is beyond the scope of this report.

In summation, Riemannian optimization provides a solid conceptual and theoretical framework for solving smooth constrained optimization problems. In particular it allows the intuitions for unconstrained minimization procedures from $\mathbb{R}^n$ to be applied to constrained problems. This is

[8]in the sense of Definition 5.4

exciting as provides a new approach to that subsumes existing theory. As a result this framework may enable new, intuitive approaches to difficult problems in both applied and theoretical research.

References

[1] Olivetti faces dataset. https://cs.nyu.edu/%7Eroweis/data.html.

[2] P-A Absil, Robert Mahony, and Rodolphe Sepulchre. *Optimization algorithms on matrix manifolds*. Princeton University Press, 2009.

[3] N. Boumal, B. Mishra, P.-A. Absil, and R. Sepulchre. Manopt, a Matlab toolbox for optimization on manifolds. *Journal of Machine Learning Research*, 15:1455–1459, 2014.

[4] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[5] Reshad Hosseini and Suvrit Sra. Manifold optimization for gaussian mixture models. *arXiv preprint arXiv:1506.07677*, 2015.

[6] Raghunandan H Keshavan and Sewoong Oh. A gradient descent algorithm on the grassman manifold for matrix completion. *arXiv preprint arXiv:0910.5260*, 2009.

[7] Artiom Kovnatsky, Klaus Glashoff, and Michael M Bronstein. Madmm: a generic algorithm for non-smooth optimization on manifolds. In *European Conference on Computer Vision*, pages 680–696. Springer, 2016.

[8] David M Rosen, Luca Carlone, Afonso S Bandeira, and John J Leonard. Se-sync: A certifiably correct algorithm for synchronization over the special euclidean group. *arXiv preprint arXiv:1612.07386*, 2016.

[9] Peter H Schönemann. A generalized solution of the orthogonal procrustes problem. *Psychometrika*, 31(1):1–10, 1966.

[10] James Townsend, Niklas Koep, and Sebastian Weichwald. Pymanopt: A python toolbox for optimization on manifolds using automatic differentiation. *Journal of Machine Learning Research*, 17(137):1–5, 2016.

A Code Listings

```
function [r] = basicRetract(x,y)



r= x+y;
r = r/norm(r);

end
```

Listing 1: Basic retraction for optimization on S^{n-1}

```
function [r] = exmapRetract(x,y)

r = x*cos(norm(y)) + (y/norm(y))*sin(norm(y));
end
```

Listing 2: Exponential map retraction for optimization on S^{n-1}

```
function [x,gaps] = rayleighLS(retract,costfun, x, A,alpha, beta, sigma,p_star,
    max_iters,tol)
% ASSUME C=1

iters = 0;
p = costfun(x);
gap = abs(p_star-p);
gaps = [gap]
while iters<max_iters && gap >=tol
    eta = -2*(A*x-x*x'*A*x);
    s = beta;
    while costfun(retract(x,alpha*s*eta))>costfun(x)-sigma*alpha*s*eta'*eta
        s = s*beta;
    end
    y = alpha*s*eta;
    x = retract(x,y);
    p = costfun(x);
    gap = abs(p_star-p);
    gaps = [gaps, gap];
    iters = iters+1;
end
```

Listing 3: Line Search Algorithm for minimizing Equation (A)

```
function[c] = rayleighCost(x,A)

c = x'*A*x;
end
```

Listing 4: Cost function for Equation (A)

```
format long e
diag_sizes=[2,10,50,100]
%{

ns = [2,5,20,50,100];
n=100;
alpha = 1;
beta = 0.5;
sigma = 0.5;
A = rand(n,n); %make a random symmetric matrix
A=A*A'


A=diag(linspace(1,n,n))
cfun = @(x)rayleighCost(x,A)

x0 = rand(n,1); %% intialize x0
x0 = x0/norm(x0);

rfun = @(x,y)basicRetract(x,y)
maxit = 1000;
tol = 10E-8;

p_star = eig(A);
p_star = p_star(1);


%[x_star, gaps] = rayleighLS(rfun, cfun, x0,A, alpha, beta, sigma,p_star,
%maxit,tol);
```

```
rfunB = @(x,y)basicRetract(x,y);
rfunE = @(x,y)exmapRetract(x,y);

ns = [2,5,20,50,100,1000];
labels = [];
figure;
for n=ns

    % set up labels for legend
    lab = string(sprintf('n = %d',n));
    labels = [labels, lab];





    x0 = rand(n,1); % initialize unit norm init point
    x0 = x0/norm(x0);

    A = rand(n,n); %make a random symmetric matrix
    A=A*A';

    p_star = eig(A);
    p_star = p_star(1);


    cfun = @(x)rayleighCost(x,A); % intialize the cost function


    [x_star, gaps] = rayleighLS(rfunB, cfun, x0,A, alpha, beta, sigma,p_star, maxit
    ,tol);
    semilogy(gaps,'LineWidth',2);
    hold on;
end
legend(labels)
title({'Minimizing Rayleigh Quotient on Sphere for nxn Symmetric Matrices','
    Gradient Descent on Manifold S^{n-1}'})
xlabel('Iterations')
ylabel('Optimality Gap')
fig = gcf;
fig.PaperPositionMode = 'auto'
fig_pos = fig.PaperPosition;
fig.PaperSize = [fig_pos(3) fig_pos(4)];
print(fig,'rayleigh1.pdf','-dpdf')
close(gcf)


ns = [2,10,50,1e2,500,1e3];
labels = [];
figure;


for n=ns

    % set up labels for legend
    lab = string(sprintf('n = %d',n));
    labels = [labels, lab];



    % intialize a

    x0 = rand(n,1); % initialize unit norm init point
    x0 = x0/norm(x0);

    %A = rand(n,n); %make a random symmetric matrix
    %A=A*A';
    A=diag(1:n)
    p_star = eig(A);
    p_star = p_star(1);


    cfun = @(x)rayleighCost(x,A); % intialize the cost function


    [x_star, gaps] = rayleighLS(rfunB, cfun, x0,A, alpha, beta, sigma,p_star, maxit
    ,tol);
    semilogy(gaps,'LineWidth',2);
    hold on;
end
legend(labels)
title({'Minimization Comparison for Diag(1,...,n)','Gradient Descent on Manifold S
    ^{n-1}'})
xlabel('Iterations')
ylabel('Optimality Gap')
fig = gcf;
fig.PaperPositionMode = 'auto'
fig_pos = fig.PaperPosition;
fig.PaperSize = [fig_pos(3) fig_pos(4)];
print(fig,'rayleigh_unit_powersof10.pdf','-dpdf')
close(gcf)

%}

%{ retraction comparison %}
ns =[50,100,500]
retractions={rfunB, rfunE};
figure
labels = [];

tol = 10E-3;
for n=ns
    if n==50
        A= rand(n,n); % small random matrices
        A=A*A'; %small random matrices
    else
        A=diag(linspace(1,n,n)); % diagonal matrix
    end
    x0 = rand(n,1); % initialize unit norm init point
    x0 = x0/norm(x0);
    cfun = @(x)rayleighCost(x,A)
 for j=1:length(retractions)
    rfun = retractions{j};
    [x_star, gaps] = rayleighLS(rfun, cfun, x0,A, alpha, beta, sigma,p_star, maxit,
    tol);
```

```matlab

    if j==1
        semilogy(gaps,'LineWidth',2)
        hold on;
        if n==50
            lab = string(sprintf('Random %dx%d. Ret=basic',n,n));
            labels = [labels, lab];
        else
            lab = string(sprintf('Diag(1..d). Ret=basic',n));
            labels = [labels, lab];
        end
    else
        semilogy(gaps,'LineWidth',2)
        if n==50
            lab = string(sprintf('Random %dx%d. Ret=exmap',n,n));
            labels = [labels, lab];
        else
            lab = string(sprintf('Diag(1..d). Ret=exmap',n));
            labels = [labels, lab];
        end
    end
 end

end
legend(labels);
title('Comparing Retractions on Unit Sphere with Various Matrices')
xlabel('Number of Iterations')
ylabel('Optimality Gap')

fig = gcf;
fig.PaperPositionMode = 'auto'
fig_pos = fig.PaperPosition;
fig.PaperSize = [fig_pos(3) fig_pos(4)];
print(fig,'retraction_comparison.pdf','-dpdf')
close(gcf)
```

Listing 5: Code to generate Figures 1 to 3

```matlab
function [f]=faceCost(X,W,Z)

f = norm(X*W-Z,'fro');
end
```

Listing 6: Cost function for the Procrustes Problem.

```matlab
function [S] = sym(X)

S = 0.5*(X+X');
end
```

Listing 7: Function to compute $sym(X)$.

```matlab
function [X,X_e,gaps] = stieffelFaceLS(retract,costfun, X,W,Z,alpha,p_star, beta,
    sigma,max_iters)

X_e=0;
iters = 0;
gaps = [];
gap = norm(X-p_star,'fro');
gaps = [gaps,gap];
```

```
thresh = 1000;
sample_point = floor(0.3*max_iters);
while iters< max_iters
    t0 = X*W-Z;
    g = (1/norm(t0,'fro'))*t0*W';
    grad = g-X*sym(X*g);
    eta = -grad;
    s = beta;
    count = 0;
    while costfun(X+alpha*s*eta)>costfun(X)-sigma*alpha*s*trace(eta'*eta) && count<
    thresh
        s = s*beta;
        count = count+1
    end
    if iters==sample_point
        X_e=X;
    end
    disp('line search done')
    X=retract(X,alpha*s*eta);
    iters = iters+1
    gap = norm(X-p_star,'fro');
    gaps = [gaps,gap];
    X;
end
```

Listing 8: Function to perform line search from Equation (C)

```
face1 = load('face1.mat');
face2 = load('face2.mat');
face3 = load('face3.mat');
face4 = load('face4.mat');
face5 = load('face5.mat');


face1 =face1.zz
face2 = face2.zz
face3 = face3.zz
face4 = face4.zz
face5 = face5.zz


% global attributes
alpha = 1;
beta=0.5;
sigma=0.5;
max_iters=1000;
n=64;


A = rand(n,n);
[R,Q]=qr(A'*A);

W = R*face1;% rotated face
Z = face1;

B=rand(n,n);
[X0,~] = qr(B'*B);
retfun = @(x,y)stieffelRetract(x,y);
costfun= @(X)faceCost(X,W,Z);
costfun(inv(R))
```

```
[i,j,k] =svd(Z*W');
p_star = i*k';
%X0=inv(R)
[x,X_e,gap]=stieffelFaceLS(retfun, costfun, X0,W,Z,alpha,p_star,beta,sigma,
    max_iters)

figure
colormap(gray);
subplot(1,4,1);
imagesc(face1);
subplot(1,4,2);
imagesc(W);
subplot(1,4,3);
imagesc(X_e*W);
subplot(1,4,4);
imagesc(x*W);
fig = gcf;
fig.PaperPositionMode = 'auto'
fig_pos = fig.PaperPosition;
fig.PaperSize = [fig_pos(3) fig_pos(4)];
print(fig,'Faces','-dpdf')
close(gcf)

%semilogy(gap)


imagesc(x)
fig = gcf;
fig.PaperPositionMode = 'auto'
fig_pos = fig.PaperPosition;
fig.PaperSize = [fig_pos(3) fig_pos(4)];
print(fig,'learnedd_matrix','-dpdf')
close(gcf)
```

Listing 9: Function to generate Figure 4.