ManifoldOptim: An R Interface to the ROPTLIB Library for Riemannian Manifold Optimization

Sean Martin
UMBC

Andrew M. Raim
U.S. Census Bureau

Wen Huang
Florida State University

Kofi P. Adragni
UMBC

Abstract

Manifold optimization appears in a wide variety of computational problems in the applied sciences. In recent statistical methodologies such as sufficient dimension reduction and regression envelopes, estimation relies on the optimization of likelihood functions over spaces of matrices such as the Stiefel or Grassmann manifolds. Recently, Huang, Absil, Gallivan, and Hand (2016) have introduced the library **ROPTLIB**, which provides a framework and state of the art algorithms to optimize real-valued objective functions over commonly used matrix-valued Riemannian manifolds. This article presents **ManifoldOptim**, an R package that wraps the C++ library **ROPTLIB**. **ManifoldOptim** enables users to access functionality in **ROPTLIB** through R so that optimization problems can easily be constructed, solved, and integrated into larger R codes. Computationally intensive problems can be programmed with **Rcpp** and **RcppArmadillo**, and otherwise accessed through R. We illustrate the practical use of **ManifoldOptim** through several motivating examples involving dimension reduction and envelope methods in regression.

Keywords: Manifold, Riemannian, Grassmann, Stiefel, Euclidean, Optimization.

1. Introduction

This article presents **ManifoldOptim**, an R (R Core Team 2016) package for the optimization of real-valued functions on Riemannian manifolds. **ManifoldOptim** is a wrapper for **ROPTLIB**, a C++ optimization library by Huang *et al.* (2016), which provides a variety of cutting edge algorithms for optimization on manifolds and a framework for constructing such optimization

problems.

There is a growing literature on manifolds and manifold optimization. Manifolds are generalizations of smooth curves and surfaces within higher dimensions. Riemannian manifolds are a class of manifolds which have rules to compute distances and angles. Some manifolds are especially abundant in the literature because of their wide applicability; for example, the differential geometry of Grassmann manifolds has been well studied (Wong 1967; Edelman *et al.* 1998; Chikuse 2003; Absil *et al.* 2004). Mathematical introductions to manifolds can be found in Lee (2000, 2003) and Tu (2011).

Optimization on Riemannian manifolds generalizes Euclidean optimization algorithms to curved and smoothed surfaces (Absil *et al.* 2004; Wong 1967; Edelman *et al.* 1998; Chikuse 2003). Applications are found in many areas including signal processing (Comon and Golub 1990), data mining (Berry *et al.* 1999), and control theory (Patel *et al.* 1994). Many eigenvalue problems are, in fact, optimization problems on manifolds (Absil *et al.* 2004; Edelman *et al.* 1998). Recent developments in likelihood-based sufficient dimension reduction (Cook and Forzani 2008; Cook and Li 2009; Cook and Forzani 2009) and regression envelope models (Cook *et al.* 2010; Su and Cook 2011, 2012; Cook and Su 2013; Cook and Zhang 2015) rely on optimizations over manifolds.

Formulating optimization algorithms on manifolds requires endowing the manifolds with a differentiable structure. The gradient and Hessian are often used in Euclidean optimization to find the direction for the next iterate, but these notions must be extended to honor the curved structure of the manifold within Euclidean space. Each manifold is unique with respect to its differential geometry and requires some individual consideration. In this article, we focus on the following manifolds: Euclidean, Grassmann, low-rank manifold, manifold of n-dimensional vectors on the unit sphere, Stiefel, and space of semi-positive definite matrices. We also consider spaces formed by taking Cartesian products of these manifolds. Available optimization algorithms include symmetric rank-one update (Huang *et al.* 2015a), limited-memory (Huang *et al.* 2015a), steepest descent (Absil *et al.* 2004), line-search Newton (Absil *et al.* 2004), conjugate gradients (Nocedal and Wright 1999; Absil *et al.* 2004; Sato and Iwai 2013), and steepest descent (Absil *et al.* 2004).

Consider the problem of minimizing $f(\mathrm{U})$ for $\mathrm{U} \in \mathcal{M}$ where $\mathcal{M}$ is a Riemannian manifold. For example, let U be a $p \times d$ semi-orthogonal matrix where $\mathrm{U}^T\mathrm{U} = \mathrm{I}_d$ and $d < p$. This problem naturally lies in a Stiefel manifold, where orthonormality of the argument U is intrinsic to the manifold. When f is invariant under right orthogonal transformations of U, so that $f(\mathrm{U}) = f(\mathrm{U}\mathrm{O})$ for any $d \times d$ orthogonal matrix O, then the problem lies in a Grassmann manifold. If U is a $p \times p$ symmetric positive definite (SPD) matrix, such as a covariance matrix, the problem is naturally defined on the manifold of SPD matrices. Methods abound in the literature for optimization problems with linear constraints (Fletcher 1987; Nocedal and Wright 1999). Constrained problems sometimes become unconstrained by an appropriate transformation. The constraints in the above examples are naturally honored by restricting to the respective manifolds.

In a typical use of **ManifoldOptim**, the user provides an objective function, its gradient and Hessian, and a specification of the manifold, the solver, and the optimization method to be used. Numerical gradient and Hessian functions are provided, as defaults, for problems where closed-form expressions are not easily programmed. **ManifoldOptim** users can construct problems in plain R for convenience and quick prototyping. If performance becomes a concern,

users can implement the objective, gradient, and Hessian functions in C++ and otherwise interact with **ManifoldOptim** through R. We make use of the **Rcpp** (Eddelbuettel and Francois 2011) and **RcppArmadillo** (Eddelbuettel and Sanderson 2014) packages to reduce the burden of C++ programming for R users, and to facilitate seamless integration with R. The **ROPTLIB** library itself is written in C++ and uses standard linear algebra libraries such as **BLAS** and **LAPACK** (Anderson *et al.* 1999) to ensure generally good performance.

ROPTLIB can be readily used in Matlab through its included `mex`-based wrapper. Interfaces to **ROPTLIB** for other high level languages, such as Python and Julia, have also been developed (Huang *et al.* 2016). Other manifold optimization software packages are found in the literature: **sg_min** written by Lippert (2000) which was adapted from Edelman *et al.* (1998), and **Manopt** of Boumal *et al.* (2014) are available in Matlab. In R, Adragni and Wu (2010) developed **GrassmannOptim** specifically for the Grassmann manifold. To our knowledge, there are no other publicly available R packages for manifold optimization.

The following notations are adopted throughout this manuscript: $\mathcal{G}_{d,p}$ represents the Grassmann manifold, the set of all d-dimensional subspaces in $\mathbb{R}^p$; $\mathcal{S}_{d,p}$ represents the Stiefel manifold, the set of all d-dimensional orthonormal matrices in $\mathbb{R}^p$; $\mathcal{S}_p^+$ represents the manifold of all $p \times p$ symmetric positive definite matrices, and $\mathcal{S}^p$ is the unit sphere manifold.

The remainder of this article is organized as follows. Section 2 gives a brief description of the algorithms that guided the programming of **ManifoldOptim**. Section 3 demonstrates usage of the package through brief examples. Section 4 presents several applications where manifold optimization is useful in statistics. R code for these applications is somewhat lengthy and therefore is provided as supplementary material. Discussions and conclusions follow in section 5. The **ManifoldOptim** package is available from the Comprehensive R Archive Network at `http://cran.r-project.org`.

2. Optimization on Manifolds

Manifold optimization appears in a wide variety of computational problems in the applied sciences. It has recently gained prominence in the statistics literature. Optimization over a Grassmann manifold is used in several models for sufficient dimension reduction (SDR) in regression (Cook 1998, 2007), including covariance reducing models (Cook and Forzani 2008) and likelihood acquired directions (Cook and Forzani 2009). Adragni and Wu (2010) provide some relevant examples. Essentially, a manifold optimization is characterized by symmetry or invariance properties of the objective function.

Consider the Brockett problem (Absil *et al.* 2004, section 4), which amounts to minimizing the objective function

$$f(X) = \text{Trace}(X^T B X D), \quad X \in \mathbb{R}^{p \times d} \tag{1}$$

subject to $X^T X = I_p$. Here, $D = \text{Diag}(\mu_1, \ldots, \mu_p)$ with $\mu_1 \geq \cdots \geq \mu_p \geq 0$ and $B \in \mathbb{R}^{n \times n}$ is a given symmetric matrix. The optimization can be carried out over a Stiefel manifold $\mathcal{S}_{d,p}$. It is known that a global minimizer of f is the matrix whose columns are the eigenvectors of B corresponding to the d smallest eigenvalues $\mu_{p-d+1}, \ldots, \mu_p$. This is essentially an eigenvalue problem reminiscent of the generalized Rayleigh quotient for discriminant analysis (Adragni and Wu 2010). It will be later used in section 3 to illustrate practical use of the package.

We next present three algorithms in statistics that can be formulated as manifold optimization problems. The first is a recently developed Minimum Average Deviance Estimation method

for SDR (Adragni *et al.* 2016). Second is Cook's Principal Fitted Components model for SDR (Cook 2007). Third is an envelope method for regression initially proposed by Cook *et al.* (2010). These problems will be explored further in section 4.

2.1. Minimum Average Deviance Estimation

Minimum Average Deviance Estimation (MADE) is a dimension reduction method based on local regression (Adragni *et al.* 2016). This approach was first proposed by Xia *et al.* (2002) under the assumption of additive errors. Suppose $Y \in \mathbb{R}$ is a response, X is a p-dimensional predictor, and the distribution of $Y \mid X$ is given by an exponential family distribution of the form

$$f(Y \mid \theta(X)) = f_0(Y, \phi) \exp \left\{ \frac{Y \cdot \theta(X) - b(\theta(X))}{a(\phi)} \right\} \tag{2}$$

with dispersion parameter ϕ. The canonical parameter $\theta(X)$ possesses the main information that connects Y to X; it relates to the mean function $E(Y \mid X)$ through a link function g so that $g(E(Y \mid X)) = \theta(X)$. Let (Y_i, X_i), $i = 1, \ldots, n$, represent an independent sample from the distribution of (Y, X) so that $Y_i \mid X_i$ has the distribution (2). If there exists a semi-orthogonal $B \in \mathbb{R}^{p \times d}$ with $d < p$ and $\theta(X) = \vartheta(B^T X)$ for some function ϑ, then $X \in \mathbb{R}^p$ can be replaced by $B^T X \in \mathbb{R}^d$ in the regression of Y on X, and the subspace $\mathcal{S}_B$ is a dimension reduction subspace.

Regression based on the local log-likelihood evaluated at a given $X_0 \in \mathbb{R}^p$ can be written as

$$L_{X_0}(\alpha_0, \gamma_0, B) = \sum_{i=1}^n w(X_i, X_0) \log f(Y_i \mid \alpha_0 + \gamma_0^T B^T (X_i - X_0)),$$

using the first-order Taylor expansion

$$\vartheta(B^T X_i) \approx \vartheta(B^T X_0) + [\nabla \vartheta(B^T X_0)]^T (B^T X_i - B^T X_0)$$
$$= \alpha_0 + \gamma_0^T B^T (X_i - X_0),$$

taking $\alpha_0 = \vartheta(B^T X_0)$ and $\gamma_0 = \nabla \vartheta(B^T X_0)$. The 0 subscript in α_0, γ_0 denotes that the parameters vary with the choice of X_0. The weights $w(X_1, X_0), \ldots, w(X_n, X_0)$ represent the contribution of each observation toward $L_{X_0}(\alpha_0, \gamma_0, B)$. A local deviance for the jth observation can be defined as

$$D(Y_j, \vartheta(B^T X_j)) = 2 \left[\max_{\vartheta} \log f(Y_j \mid \vartheta) - L_{X_j}(\alpha_j, \gamma_j, B) \right]. \tag{3}$$

The term $\max_{\vartheta} \log f(Y_j \mid \vartheta)$ is the maximum likelihood achievable for an individual observation. Consider minimizing the average deviance $n^{-1} \sum_{j=1}^n D(Y_j, \vartheta(B^T X_j))$ with respect to $(\alpha_j, \gamma_j) \in \mathbb{R}^{d+1}$ for $j = 1, \ldots, n$ and $B \in \mathbb{R}^{p \times d}$ such that $B^T B = I$. This is equivalent to maximizing

$$Q(\boldsymbol{\alpha}, \boldsymbol{\gamma}, B) = \sum_{j=1}^n \sum_{i=1}^n w(X_i, X_j) \left\{ Y_i(\alpha_j + \gamma_j^T B^T (X_i - X_j)) - b(\alpha_j + \gamma_j^T B^T (X_i - X_j)) \right\} \tag{4}$$

over the same space. The kernel weight function is taken to be

$$w(X_i, X_0) = \frac{K_{\mathrm{H}}(X_i - X_0)}{\sum_{j=1}^n K_{\mathrm{H}}(X_j - X_0)}, \quad K_{\mathrm{H}}(\mathrm{u}) = |\mathrm{H}|^{-1} K(\mathrm{H}^{-1/2} \mathrm{u}),$$

where $K(u)$ denotes one of the usual multidimensional kernel density functions and the bandwidth H is a $p \times p$ symmetric and positive definite matrix. We may also consider refined weights $w(B^T X_i, B^T X_0)$ which make use of the unknown B.

For any orthogonal matrix $O \in \mathbb{R}^{d \times d}$, $\gamma^T B^T = \gamma^T O O^T B^T$, which implies that γ and B are not uniquely determined but obtained up to an orthogonal transformation. Furthermore, refined weights based on the Gaussian kernel with $H = hI$ depend on B only through $BB^T = BOO^T B^T$. In this setting, the MADE problem is invariant to orthogonal transformation of B in the sense that

$$Q(\boldsymbol{\alpha}, \boldsymbol{\gamma}, B) = Q(\boldsymbol{\alpha}, O^T \gamma_1, \ldots, O^T \gamma_n, BO).$$

The joint parameter space of $(\boldsymbol{\alpha}, \boldsymbol{\gamma}, B)$ is a product manifold $\mathbb{R}^{n(d+1)} \times \mathcal{G}_{d,p}$. However, in their initial work, Adragni *et al.* (2016) used an iterative algorithm to maximize the objective function (4):

1. Provide an initial B.

2. Do until convergence:

   (a) Fix B and maximize over $(\alpha_j, \gamma_j) \in \mathbb{R}^{d+1}$ using Newton-Raphson for $j = 1, \ldots, n$.

   (b) Fix $\boldsymbol{\alpha}$ and $\boldsymbol{\gamma}$ and the weights w_{ij}, and maximize over $B \in \mathcal{S}_{d,p}$.

   (c) Update the weights $w_{ij} = w(B^T X_i, B^T X_j)$ if refined weights are desired.

2.2. Principal Fitted Components

The principal fitted components (PFC) model was initially proposed by Cook (2007) as a likelihood-based method for sufficient dimension reduction in regression. Let X be a p-vector of random predictors and Y be the response. Cook (2007), using the stochastic nature of the predictors, proposed the model

$$\mathrm{X}_y = \mu + \Gamma \beta \mathrm{f}_y + \Delta^{1/2} \varepsilon. \tag{5}$$

Here, X_y denotes the conditional X given $Y = y$, $\mathrm{f}_y \in \mathbb{R}^r$ is a user-selected function of the response, which helps capture the dependency of X on Y. The other parameters are $\mu \in \mathbb{R}^p$, $\Gamma \in \mathbb{R}^{p \times d}$ is a semi-orthogonal matrix and $\beta \in \mathbb{R}^{d \times r}$. The error term $\varepsilon \in \mathbb{R}^p$ is assumed to be normally distributed with mean 0 and variance Δ.

Essentially, this model can be seen as a way to partition the information in the predictors given the response into two parts: one part $\Gamma \beta \mathrm{f}_y = \mathrm{E}(\mathrm{X}_y - \mathrm{X})$ that is related to the response and another part $\mu + \varepsilon$ that is not. The term that is related to y suggests that the translated conditional means $\mathrm{E}(\mathrm{X}_y - \mathrm{X})$ fall in the d-dimensional subspace $\mathcal{S}_\Gamma$.

Cook (2007) showed that a sufficient reduction of X is $\eta^T \mathrm{X}$, where $\eta = \Delta^{-1} \Gamma$. Thus, X can be replaced by $\eta^T \mathrm{X}$ without loss of information about the regression of Y on X. However, as $\eta^T \mathrm{X}$ is a sufficient reduction, for any $d \times d$ orthogonal matrix O, $\mathrm{O}^T \eta^T \mathrm{X}$ is also sufficient. Thus Γ is not estimable but the subspace spanned by its columns is estimable. The goal of an analysis is then to estimate the subspace $\mathcal{S}_\Gamma$ spanned by the columns of Γ.

The estimation of $\mathcal{S}_\Gamma$ depends on the structure of Δ. Various structures for Δ can be modeled, including isotropic $\Delta = \sigma^2 \mathrm{I}$, diagonal $\Delta = \mathrm{Diag}(\sigma_1^2, ..., \sigma_p^2)$, the heterogeneous structure $\Delta = \Gamma \Omega \Gamma^T + \Gamma_0 \Omega_0 \Gamma_0^T$, and the general unstructured $\Delta > 0$ cases. In the heterogeneous

structure of Δ, Γ_0 is the orthogonal completion of Γ such that (Γ, Γ_0) is a $p \times p$ orthogonal matrix. The matrices $\Omega \in \mathbb{R}^{d \times d}$ and $\Omega_0 \in \mathbb{R}^{(p-d) \times (p-d)}$ are assumed to be symmetric and full-rank.

We now focus on the last two structures of Δ, the heterogeneous and the general unstructured, with the aim of their maximum likelihood estimation. Assuming that a set of n data points is observed, let $\bar{\mathrm{X}}$ be the sample mean of X and let $\mathbb{X}$ denote the $n \times p$ matrix with rows $(\mathrm{X}_y - \bar{\mathrm{X}})^T$. Let $\mathbb{F}$ denote the $n \times r$ matrix with rows $(\mathrm{f}_y - \bar{\mathrm{f}})^T$ and set $\mathrm{P}_{\mathbb{F}}$ to denote the linear operator that projects onto the subspace spanned by the columns of $\mathbb{F}$. Also let $\widehat{\mathbb{X}} = \mathrm{P}_{\mathbb{F}} \mathbb{X}$ denote the $n \times p$ matrix of the fitted values from the multivariate linear regression of X on f_y. Let $\widehat{\Sigma} = \mathbb{X}^T \mathbb{X} / n$ and $\widehat{\Sigma}_{\mathrm{fit}} = \widehat{\mathbb{X}}^T \widehat{\mathbb{X}} / n$. For the heterogeneous and the unstructured Δ, the objective functions are respectively

$$\mathcal{L}(\Omega, \Omega_0, \Gamma) = -\frac{n}{2} \log |\Omega| - \frac{n}{2} \log |\Omega_0| - \mathrm{Trace}\left\{ \Gamma^T \widehat{\Sigma}_{\mathrm{res}} \Gamma \Omega^{-1} - \Gamma_0^T \widehat{\Sigma} \Gamma_0 \Omega_0^{-1} \right\} \qquad (6)$$

$$\mathcal{L}(\Delta, \Gamma) = -\frac{n}{2} \log |\Delta| - \frac{n}{2} \mathrm{Trace}\left\{ \left[\widehat{\Sigma} - \widehat{\Sigma}_{\mathrm{fit}} \Delta^{-1} \Gamma (\Gamma^T \Delta^{-1} \Gamma)^{-1} \Gamma^T \right] \Delta^{-1} \right\}. \qquad (7)$$

The maximum likelihood estimators are obtained for the heterogeneous and unstructured Δ respectively as

$$(\widehat{\Omega}, \widehat{\Omega}_0, \widehat{\mathcal{S}}_\Gamma) = \underset{(\Omega, \Omega_0, \mathcal{S}_\Gamma) \in \mathcal{S}_d^+ \times \mathcal{S}_{p-d}^+ \times \mathcal{G}_{d,p}}{\arg\max} f(\Omega, \Omega_0, \Gamma), \qquad (8)$$

$$(\widehat{\Delta}, \widehat{\mathcal{S}}_\Gamma) = \underset{(\Delta, \mathcal{S}_\Gamma) \in \mathcal{S}_p^+ \times \mathcal{G}_{d,p}}{\arg\max} f(\Delta, \Gamma), \qquad (9)$$

The objective functions are real-valued. The optimizations are carried over product manifolds $\mathcal{S}_d^+ \times \mathcal{S}_{p-d}^+ \times \mathcal{G}_{d,p}$ and $\mathcal{S}_p^+ \times \mathcal{G}_{d,p}$.

In the two optimizations (8) and (9), the parameters are constrained to different manifolds. A common approach to maximum likelihood estimation as in this setting is to optimize one subset of parameters at a time while keeping the others fixed, cycling through the subsets until convergence. For these situations **ManifoldOptim** supports a product space of manifolds, composed from simpler manifolds which can include Euclidean spaces. Here, optimization algorithms work over all parameters jointly rather than cycling through partial optimizations. In our experience joint optimization has produced comparable, if not superior, results to cycling, with a reduced programming burden.

2.3. Envelope Method for Regression

Enveloping is a novel and nascent method initially introduced by Cook *et al.* (2010) that has the potential to produce substantial gains in efficiency for multivariate analysis. The initial development of the envelope method was in terms of the standard multivariate linear model

$$Y = \mu + \beta X + \varepsilon, \qquad (10)$$

where $\mu \in \mathbb{R}^r$, the random response $Y \in \mathbb{R}^r$, the fixed predictor vector $X \in \mathbb{R}^p$ is centered to have sample mean zero, and the error vector $\varepsilon \sim N(0, \Sigma)$. In this context some linear combinations of Y are immaterial to the regression because their distribution does not depend on X, while other linear combinations of Y do depend on X and are thus material to the

regression. In effect, envelopes separate the material and immaterial parts of Y, and thereby allow for gains in efficiency.

Suppose that we can find a subspace $\mathcal{S} \in \mathbb{R}^r$ so that

$$Q_\mathcal{S} Y \mid X \sim Q_\mathcal{S} Y \quad \text{and} \quad Q_\mathcal{S} Y \perp\!\!\!\perp P_\mathcal{S} Y \mid X, \tag{11}$$

where $\sim$ means identically distributed, $P_\mathcal{S}$ projects onto the subspace $\mathcal{S}$ and $Q = I_r - P$. For any $\mathcal{S}$ with those properties, $P_\mathcal{S} Y$ carries all of the material information and perhaps some immaterial information, while $Q_\mathcal{S}$ carries just immaterial information. Denoting $\mathcal{B} := \text{span}(\beta)$, expressions (11) hold if and only if $\mathcal{B} \subseteq \mathcal{S}$ and $\Sigma = \Sigma_\mathcal{S} + \Sigma_{\mathcal{S}^\perp}$, where $\Sigma_\mathcal{S} = \text{var}(P_\mathcal{S} Y)$ and $\Sigma_{\mathcal{S}^\perp} = \text{var}(Q_\mathcal{S} Y)$. The subspace $\mathcal{S}$ is not necessarily unique nor minimal, because there may be infinitely many subspaces that satisfy these relations in a particular problem. Cook *et al.* (2010) showed that $\mathcal{S}$ is a reducing subspace of Σ if and only if $\Sigma = \Sigma_\mathcal{S} + \Sigma_{\mathcal{S}^\perp}$, and defined the minimal subspace to be the intersection of all reducing subspaces of Σ that contain $\mathcal{B}$, which is called the Σ-envelope of $\mathcal{B}$ and denoted as $\mathcal{E}_\Sigma(\mathcal{B})$. Let $u = \dim\{\mathcal{E}_\Sigma(\mathcal{B})\}$. Then

$$\mathcal{B} \subseteq \mathcal{E}_\Sigma(\mathcal{B}) \quad \text{and} \quad \Sigma = \Sigma_\mathcal{S} + \Sigma_{\mathcal{S}^\perp},$$

where $\mathcal{E}_\Sigma(\mathcal{B})$ is shortened to $\mathcal{E}$ for subscripts. These relationships establish a unique link between the coefficient matrix β and the covariance matrix Σ of model (10), and it is this link that has the potential to produce gains in the efficiency of estimates of β relative to the standard estimator of β. Su and Cook (2011), Su and Cook (2012), and Cook and Su (2013) among others expanded the applications of this envelope method.

Cook and Zhang (2015) recently further expanded the envelope method to generalized linear regression models. The response belongs to an exponential family of the form (2). Let $\mathcal{C}(\vartheta) = y\vartheta - b(\vartheta)$ and $W(\vartheta) = \mathcal{C}''(\vartheta)/E(\mathcal{C}''(\vartheta))$. They reparameterized $\vartheta(\alpha, \beta) = \alpha + \beta^T X$ to $\vartheta(\alpha, \beta) = a + \beta^T \{X - E(WX)\}$ with $a = \alpha + \beta^T E(WX)$ so that a and β are orthogonal. The asymptotic variance of $\hat{\beta}$ is then obtained as $avar(\sqrt{n}\hat{\beta}) = \mathbf{V}_{\beta\beta}(a, \beta) = \{E(-\mathcal{C}'' E\{[X - E(WX)][X - E(WX)]^T\}\}$. With a reparameterization $\beta = \Gamma\eta$, the parameter η is estimated using a Fisher scoring method fitting the GLM of Y on $\Gamma^T X$. The partially maximized log-likelihood for Γ is

$$L_n(\Gamma) = \sum_{i=1}^n \mathcal{C}(\alpha + \hat{\eta}^T \Gamma^T X_i) - \frac{n}{2}\{\log|\Gamma^T S_X \Gamma| + \log|\Gamma^T S_X^{-1} \Gamma| + \log|S_X|\},$$

where S_X is the sample covariance of X. This function is now optimized over a Grassmann manifold to obtain $\widehat{\Gamma}$.

3. Using ManifoldOptim

We now describe use of the **ManifoldOptim** package. The core function is `manifold.optim`, which has the following interface.

```
manifold.optim(prob, mani.defn, method = "LRBFGS", x0 = NULL,
    has.hhr = FALSE, mani.params = get.manifold.params(),
    H0 = NULL, solver.params = get.solver.params())
```

Table 1: List of optimization algorithms available in the `ManifoldOptim` package.

Code	Description
`RTRNewton`	Riemannian trust-region Newton (Absil *et al.* 2007)
`RTRSR1`	Riemannian trust-region symmetric rank-one update (Huang *et al.* 2015a)
`LRTRSR1`	Limited-memory (Huang *et al.* 2015a)
`RTRSD`	Riemannian trust-region steepest descent (Absil *et al.* 2008)
`RNewton`	Riemannian line-search Newton (Absil *et al.* 2008)
`RBroydenFamily`	Riemannian Broyden family (Huang *et al.* 2015b)
`RWRBFGS` and `RBFGS`	Riemannian BFGS (Ring and Wirth 2012; Huang *et al.* 2015b)
`LRBFGS`	Limited-memory RBFGS (Huang *et al.* 2015b)
`RCG`	Riemannian conjugate gradients (Absil *et al.* 2008; Sato and Iwai 2013)
`RSD`	Riemannian steepest descent (Absil *et al.* 2008)

A typical call to `manifold.optim` involves four essential pieces: a problem `prob`, the choice of manifold `mani.defn`, configuration for the manifold `mani.params`, and configuration for the solver `solver.params`.

The `problem` encapsulates the objective function to be minimized, and, optionally, the gradient and Hessian functions. Analytical expressions of the gradient and Hessian are usually desirable but may be either tedious to compute or intractable. Numerical approximations to the gradient and Hessian via finite differences are used by default. There are three options for constructing a problem with **ManifoldOptim**: RProblem, VectorManifoldOptimProblem, and MatrixManifoldOptimProblem.

Generally, **ManifoldOptim** treats the optimization variable as a single vector, and the user must reshape the vector into the matrices and vectors specific to the problem at hand. This approach is similarly taken by the standard `optim` function in the R package **stats** (R Core Team 2016).

The `method` argument specifies the algorithm to be used in the optimization; LRBFGS is the default method. A list of possible values for `method` is provided in Table 1.

ROPTLIB provides solvers for at least nine commonly encountered manifolds at the time of this writing. In the first version of **ManifoldOptim**, we have focused on six manifolds: unconstrained Euclidean space, the Stiefel manifold, the Grassmann manifold, the unit sphere, the orthogonal group, and the manifold of symmetric positive definite matrices. As **ROPTLIB** continues development, and as the need arises in the R community, support for more solvers and manifolds will be added to **ManifoldOptim**.

3.1. Solving the Brockett Problem

We revisit the Brockett problem described in section 2 to illustrate the use of **ManifoldOptim**. The gradient for the objective function can be obtained in closed form as

$$\nabla f(X) = 2BXD.$$

Denoting $\otimes$ as the Kronecker product operator, we may write $\text{vec}(\nabla f(X)) = 2(D \otimes B)\text{vec}(X)$ to obtain the Hessian $\nabla^2 f(\text{vec}(X)) = 2(D \otimes B)$. It should be noted that the usual Euclidean gradient and Hessian are to be computed here, ignoring manifold constraints. The **ROPTLIB** library can also work with the Riemannian gradient, but this is not exposed in **ManifoldOptim**.

We set the matrices B and D to prepare a concrete instance of the problem.

```
set.seed(1234)
n <- 150; p <- 5
B <- matrix(rnorm(n*n), nrow=n)
B <- B + t(B)    # Make symmetric
D <- diag(p:1, p)
```

The objective and gradient are coded in R as follows.

```
tx <- function(x) { matrix(x, n, p) }
f <- function(x) { X <- tx(x); Trace( t(X) %*% B %*% X %*% D ) }
g <- function(x) { X <- tx(x); 2 * B %*% X %*% D }
```

The `tx` function reshapes an np-dimensional point from the solver into an $n \times p$ matrix, which can then be evaluated by the `f` and `g` functions. We will initially select a solver that does not make use of a user-defined Hessian function.

We first consider an `RProblem`, where all functions are programmed in R. This provides a convenient way to code a problem, but may face some performance limitations. In general, functions coded in R tend to be much slower than similar functions written in C++, especially if they cannot be vectorized. With an `RProblem`, each evaluation of the objective, gradient, and Hessian by the solver incurs the overhead of a call from C++ to a function defined in R. Having noted the potential performance issues, we now proceed with construction of an `RProblem`.

```
mod <- Module("ManifoldOptim_module", PACKAGE = "ManifoldOptim")
prob <- new(mod$RProblem, f, g)
```

The `Module` construct (Eddelbuettel and François 2016) is useful for the internal implementation of **ManifoldOptim**. When constructing a module in R, the user implicitly creates a C++ problem which can be accessed by the C++ solver.

The Brockett function is to be optimized over a Stiefel manifold, so we use `get.stiefel.defn` to create the specification. Additionally, we set software parameters for the manifold and solver via the `get.manifold.params` and `get.solver.params` functions.

```
mani.defn <- get.stiefel.defn(n, p)
mani.params <- get.manifold.params(IsCheckParams = TRUE)
solver.params <- get.solver.params(isconvex = TRUE, DEBUG = 0,
    Tolerance = 1e-4, Max_Iteration = 1000, IsCheckParams = TRUE)
```

The argument `IsCheckParams = TRUE` requests useful information to be printed on the console for either the manifold or solver, depending where it is placed. `DEBUG` is an integer that sets the verbosity of the solver during iterations; The lowest verbosity is `DEBUG = 0`, which prints no debugging information. `Max_Iteration` and `Tolerance` set the maximum iteration and tolerance to determine when the solver will halt. More extensive descriptions for the arguments are given in the **ManifoldOptim** manual.

A starting value for the optimization can be specified by the argument `x0` in the call of the function `manifold.optim`. If available, a good initial value can assist the solver by reducing the time to find a solution and improving the quality of the solution when multiple local

optima are present. If no initial value is given, the optimizer will select one at random from the given manifold. In this case of the Brockett problem, we consider the following initial value.

```
x0 <- as.numeric(diag(n)[,1:p])
```

Now that we have specified the problem, manifold definition, software parameters for the manifold and solver, an algorithm, and an initial value, we can invoke the optimizer through the `manifold.optim` function.

```
res <- manifold.optim(prob, mani.defn, x0 = x0, method = "RTRSR1",
    mani.params = mani.params, solver.params = solver.params)
```

Upon completion, `manifold.optim` produces a result, `res`, which contains the quantities listed below.

```
> names(res)
 [1] "xopt"         "fval"         "normgf"       "normgfgf0"
 [5] "iter"         "num.obj.eval" "num.grad.eval" "nR"
 [9] "nV"           "nVp"          "nH"           "elapsed"
[13] "funSeries"    "gradSeries"   "timeSeries"
```

The **ManifoldOptim** manual gives a description of each of these items. The most important output is the solution `xopt`, which optimizes the objective function. Using `tx` to reshape it into a matrix, its first rows are given below.

```
> head(tx(round(res$xopt, digits=4)))
        [,1]     [,2]     [,3]     [,4]     [,5]
[1,]   0.1693  -0.1091  -0.0879  -0.0391  -0.0518
[2,]  -0.0084   0.1276  -0.1241   0.0483   0.1055
[3,]   0.0958  -0.0243   0.0394  -0.0008  -0.0135
[4,]  -0.0373  -0.1713  -0.0475  -0.0136   0.0660
[5,]  -0.1237  -0.0217   0.1197  -0.1235  -0.1114
[6,]  -0.0295  -0.0054   0.0509  -0.1352  -0.0131
```

Solvers such as `RNewton` (Absil *et al.* 2008) require a Hessian function to be specified for the problem. If a Hessian function is required by the solver but not provided by the user, a numerical approximation will be used. This provides a convenient default, but can be slow and potentially inaccurate. We briefly illustrate the use of a Hessian function for the Brockett `RProblem`.

Treating the optimization variable $\text{vec}(X)$ as a q-dimensional vector, the $q \times q$ Hessian function $\nabla^2 f(\text{vec}(X))$ is programmed by coding the action $[\nabla^2 f(\text{vec}(X))]\eta$ for a given η in the tangent space to the manifold at X. Further detail on the implementation of this action for either line search or trust region solver algorithms is given in Huang *et al.* (2016) and Absil *et al.* (2004). For the Brockett problem, this becomes the matrix multiplication $2(D \otimes B)\eta$. For an `RProblem`, this expression can be specified as the third argument of the constructor.

```
h <- function(x, eta) { 2 * (D %x% B) %*% eta }
prob <- new(mod$RProblem, f, g, h)
```

We note that the efficiency of the `h` function can be greatly improved by avoiding direct computation of $D \otimes B$, but proceed with this simple coding to facilitate the demonstration.

The option `IsCheckGradHess` aids in checking correctness of the gradient and Hessian. Setting this option leads to the following output, which appears to validate both the gradient and Hessian expressions.

```
> solver.params <- get.solver.params(isconvex = TRUE, IsCheckGradHess = TRUE)
> res <- manifold.optim(prob, mani.defn, method = "RNewton",
+ mani.params = mani.params, solver.params = solver.params, x0 = x0)
fx:-43.2077,fy:4.39105, fx-fy:-47.5987
i:0,|eta|:1.000e+02,(fy-fx)/<gfx,eta>:7.049e-03,(fy-fx-<gfx,eta>)/<0.5 eta,
Hessian eta>:-5.066e-02
...
i:25,|eta|:2.980e-06,(fy-fx)/<gfx,eta>:1.000e+00,(fy-fx-<gfx,eta>)/<0.5 eta,
Hessian eta>:9.905e-01
fx:-43.2077,fy:-43.2076, fx-fy:-0.000100615
...
i:34,|eta|:5.821e-09,(fy-fx)/<gfx,eta>:1.000e+00,(fy-fx-<gfx,eta>)/<0.5 eta,
Hessian eta>:1.844e+01
CHECK GRADIENT:
    Suppose the point is not a critical point.
    If there exists an interval of |eta| such that (fy - fx) / <gfx, eta>
    approximates ONE, then the gradient is probably correct!
CHECK THE ACTION OF THE HESSIAN (PRESUME GRADIENT IS CORRECT):
    Suppose the retraction is second order or the point is a critical point.
    If there exists an interval of |eta| such that (fy-fx-<gfx,eta>)/<0.5 eta,
    Hessian eta>
    approximates ONE, then the action of Hessian is probably correct.
```

3.2. Coding a Problem in C++

For optimization problems which require intensive computation or will be evaluated repeatedly, it may be worth investing some additional effort to write the problem in C++. We now illustrate by converting the Brockett problem to C++. To proceed, we extend the `VectorManifoldOptimProblem` class within **ManifoldOptim**. A `VectorManifoldOptimProblem` has objective, gradient, and Hessian functions coded in C++ using **RcppArmadillo**. The user implements this problem type by extending the abstract `VectorManifoldOptimProblem` class.

The class `BrockettProblem` is now written as follows.

```
Rcpp::sourceCpp(code = '
//[[Rcpp::depends(RcppArmadillo,ManifoldOptim)]]
#include <RcppArmadillo.h>
#include <ManifoldOptim.h>

using namespace Rcpp;
using namespace arma;
```

```cpp
class BrockettProblem : public VectorManifoldOptimProblem
{
public:
    BrockettProblem(const arma::mat& B, const arma::mat& D)
    : VectorManifoldOptimProblem(false), _B(B), _D(D) { }

    virtual ~BrockettProblem() { }

    double objFun(const arma::vec& x) const {
        arma::mat X;
        tx(X, x);
        return arma::trace(X.t() * _B * X * _D);
    }

    arma::mat gradFun(const arma::vec& x) const {
        arma::mat X;
        tx(X, x);
        return 2 * _B * X * _D;
    }

    arma::vec hessEtaFun(const arma::vec& x, const arma::vec& eta) const {
        return 2 * arma::kron(_D, _B) * eta;
    }

    void tx(arma::mat& X, const arma::vec& x) const {
        X = x;
        X.reshape(_B.n_rows, _D.n_rows);
    }

    const arma::mat& GetB() const { return _B; }
    const arma::mat& GetD() const { return _D; }

private:
    arma::mat _B;
    arma::mat _D;
};

RCPP_MODULE(Brockett_module) {
    class_<BrockettProblem>("BrockettProblem")
    .constructor<mat,mat>()
    .method("objFun", &BrockettProblem::objFun)
    .method("gradFun", &BrockettProblem::gradFun)
    .method("hessEtaFun", &BrockettProblem::hessEtaFun)
    .method("GetB", &BrockettProblem::GetB)
    .method("GetD", &BrockettProblem::GetD)
    ;
```

```
}
')
```

Note that we have used `sourceCpp` to compile the source code as a string from within R. It is also possible to write the C++ code in a standalone file and compile it using `sourceCpp`, or to have it compiled within the build of your own custom package; refer to the **Rcpp** documentation for more details. In addition to defining the `BrockettProblem` class, our source code defines a module which allows `BrockettProblem` objects to be constructed and manipulated from R. Specifically, the constructor, objective, gradient, Hessian, and accessor functions for B and D can be called from R.

The `tx` function is responsible for reshaping the vector `x` into variables which can be used to evaluate the objective, gradient, and Hessian functions. In this case, the optimization variable is a single matrix, so we simply reshape vector `x` into the matrix `X` using the **Armadillo** `reshape` function. If `x` were to contain multiple variables, the `tx` function could be modified to have multiple output arguments in its definition.

To define a `BrockettProblem` we must specify the B and D matrices via the constructor. Using the same B and D as in the previous section, we can invoke the constructor through R.

```
prob <- new(BrockettProblem, B, D)
```

Setting the manifold definition, manifold configuration, and solver configuration is done in the same way as before.

```
mani.defn <- get.stiefel.defn(n, p)
mani.params <- get.manifold.params(IsCheckParams = TRUE)
solver.params <- get.solver.params(isconvex = TRUE, DEBUG = 0,
    Tolerance = 1e-4, Max_Iteration = 1000, IsCheckParams = TRUE)
```

We use the same initial value as before. Thanks to our **Rcpp** module, we can call the problem functions from R, which makes them easier to test.

```
x0 <- as.numeric(diag(n)[,1:p])
eta <- rnorm(n*p)
prob$objFun(x0)
prob$gradFun(x0)
prob$hessEtaFun(x0, eta)
```

We may now invoke the solver.

```
res <- manifold.optim(prob, mani.defn, method = "RTRSR1",
    mani.params = mani.params, solver.params = solver.params, x0 = x0)
```

After obtaining a solution, our first task will likely be to reshape it from a vector to a matrix.

```
tx <- function(x) { matrix(x, n, p) }
head(tx(res$xopt))
```

The result itself is the same as in the previous section, and is therefore not shown here.

3.3. An Extra C++ Convenience for Functions of a Single Matrix

For problems where the optimization variable is a single matrix (e.g. optimization on a Grassmann manifold), users may extend the class `MatrixManifoldOptimProblem`. Doing this is very similar to extending `VectorManifoldOptimProblem`, but gives the convenience of having the optimization variable presented to problem functions as a matrix, partially avoiding the need for reshaping code.

Continuing with the Brockett example, the `MatrixManifoldOptimProblem` version is written as follows.

```
Rcpp::sourceCpp(code = '
//[[Rcpp::depends(RcppArmadillo,ManifoldOptim)]]
#include <RcppArmadillo.h>
#include <ManifoldOptim.h>

using namespace Rcpp;
using namespace arma;

class BrockettProblem : public MatrixManifoldOptimProblem
{
public:
    BrockettProblem(const arma::mat& B, const arma::mat& D)
    : MatrixManifoldOptimProblem(false), _B(B), _D(D) { }

    virtual ~BrockettProblem() { }

    double objFun(const arma::mat& X) const {
        return arma::trace(X.t() * _B * X * _D);
    }

    arma::mat gradFun(const arma::mat& X) const {
        return 2 * _B * X * _D;
    }

    arma::vec hessEtaFun(const arma::mat& X, const arma::vec& eta) const {
        return 2 * arma::kron(_D, _B) * eta;
    }

    const arma::mat& GetB() const { return _B; }
    const arma::mat& GetD() const { return _D; }

private:
    arma::mat _B;
    arma::mat _D;
};
```

```
RCPP_MODULE(Brockett_module) {
    class_<BrockettProblem>("BrockettProblem")
    .constructor<mat,mat>()
    .method("objFun", &BrockettProblem::objFun)
    .method("gradFun", &BrockettProblem::gradFun)
    .method("hessEtaFun", &BrockettProblem::hessEtaFun)
    .method("GetB", &BrockettProblem::GetB)
    .method("GetD", &BrockettProblem::GetD)
    ;
}
')
```

The code is very similar to Section 3.3, except now the `tx` function is no longer necessary. Note that the solver still treats the optimization variable as a vector, and the initial value `x0` is coded as a vector, but the objective, gradient, and Hessian expect a matrix. To evaluate the problem functions from R at `x0`, we can transform `x0` to a matrix using our `tx` function.

```
x0 <- as.numeric(diag(n)[,1:p])
eta <- rnorm(n*p)
tx <- function(x) { matrix(x, n, p) }
prob$objFun(tx(x0))
prob$gradFun(tx(x0))
prob$hessEtaFun(tx(x0), eta)
```

3.4. Product Manifold

A product of manifolds can be used to optimize jointly over multiple optimization variables. As a demonstration, consider observing a random sample $Y_1, \ldots, Y_n$ from a p-variate normal distribution with mean μ constrained to the unit sphere and symmetric positive definite variance Σ. The maximum likelihood estimators are obtained as

$$(\hat{\mu}, \widehat{\Sigma}) = \underset{\mu \in \mathcal{S}^p, \Sigma \in \mathcal{S}_p^+}{\arg\max} \left\{ -\frac{np}{2} \log(\pi) - \frac{n}{2} \log|\Sigma| - \frac{1}{2} \sum_{i=1}^{n} (Y_i - \mu)^T \Sigma^{-1} (Y_i - \mu) \right\}.$$

To code this problem, let us first generate an example dataset.

```
set.seed(1234)
n <- 400
p <- 3
mu.true <- rep(1/sqrt(p), p)
Sigma.true <- diag(2,p) + 0.1
y <- mvtnorm::rmvnorm(n, mean = mu.true, sigma = Sigma.true)
```

Next we define the objective, and a function `tx` which reshapes a dimension $p + p^2$ vector into a p-dimensional vector and a $p \times p$ matrix.

```
tx <- function(x) {
    list(mu = x[1:p], Sigma = matrix(x[1:p^2 + p], p, p))
}
f <- function(x) {
    par <- tx(x)
    -sum(dmvnorm(y, mean = par$mu, sigma = par$Sigma, log = TRUE))
}
mod <- Module("ManifoldOptim_module", PACKAGE = "ManifoldOptim")
prob <- new(mod$RProblem, f)
```

We have used an `RProblem` so that the objective function can be specified as an `R` function. The negative of the log-likelihood has been given as the objective to achieve a maximization. We have not specified gradient or Hessian functions so that numerical approximations will be used by the solver. A product manifold definition is now constructed for the problem using unit sphere manifold $\mathcal{S}^p$ and SPD manifold $\mathcal{S}_p^+$ definitions. We also specify options for the manifold and solver.

```
mani.defn <- get.product.defn(get.sphere.defn(p), get.spd.defn(p))
mani.params <- get.manifold.params()
solver.params <- get.solver.params(isconvex = TRUE, Tolerance = 1e-4)
```

Note that, for any unconstrained parameters, the `get.euclidean.defn` function can be used in the product of manifolds. We now give an initial value and invoke the solver.

```
mu0 <- diag(1, p)[,1]
Sigma0 <- diag(1, p)
x0 <- c(mu0, as.numeric(Sigma0))
res <- manifold.optim(prob, mani.defn, method = "LRBFGS",
    mani.params = mani.params, solver.params = solver.params, x0 = x0)
```

The following result is produced.

```
> tx(res$xopt)
$mu
[1] 0.5074111 0.7052620 0.4951155

$Sigma
            [,1]       [,2]       [,3]
[1,]  2.0019228 0.1370588 -0.1698529
[2,]  0.1370588 2.0330334  0.2678429
[3,] -0.1698529 0.2678429  2.0967738
```

The result for Σ may not be exactly symmetric due to numerical error. If exact symmetry is needed for Σ, the symmetric part can be extracted as the estimate.

```
S <- tx(res$xopt)$Sigma
Sigma.hat <- 1/2 * (S + t(S))
```

3.5. Customizing Line Search

Line search algorithms are analogous to the method of gradient descent. As described in detail in Absil *et al.* (2004), they are based on an update

$$x_{k+1} = x_k + t_k \eta_k, \tag{12}$$

where $\eta_k \in \mathbb{R}^n$ is the search direction and $t_k \in \mathbb{R}$ is the step size. Determining a step size on the curved surface of a manifold requires special consideration beyond what is needed for Euclidean space. The concept of a retraction mapping is employed to move in the direction of a tangent vector while staying on the manifold. Several line search algorithms are available in the **ROPTLIB** library, along with options to customize the search. Some of these options can be configured through the R interface.

As an example, consider the product manifold example from Section 3.4. Several line search parameters are specified below. In particular, `LineSearch_LS = 1` corresponds to the Wolfe line search method.

```
solver.params <- get.solver.params(isconvex = TRUE, Tolerance = 1e-4,
    IsCheckParams = TRUE, LineSearch_LS = 1, Accuracy = 2e-4,
    Finalstepsize = 1/2)
```

Now, running the LRBFGS optimizer results in the following output.

```
> res <- manifold.optim(prob, mani.defn, method = "LRBFGS",
    mani.params = mani.params, solver.params = solver.params, x0 = x0)
GENERAL PARAMETERS:
Stop_Criterion:       GRAD_F_0[YES],        Tolerance    :        0.0001[YES]
Max_Iteration :          1000[YES],         Min_Iteration :            0[YES]
OutputGap     :             1[YES],         DEBUG        :      NOOUTPUT[YES]
LINE SEARCH TYPE METHODS PARAMETERS:
LineSearch_LS :         WOLFE[YES],         LS_alpha     :        0.0001[YES]
LS_beta       :         0.999[YES],         Initstepsize :             1[YES]
Minstepsize   :   2.22045e-16[YES],         Maxstepsize  :          1000[YES]
Accuracy      :        0.0002[YES],         Finalstepsize :          0.5[YES]
Num_pre_funs  :             0[YES],         InitSteptype :    QUADINTMOD[YES]
LRBFGS METHOD PARAMETERS:
nu            :        0.0001[YES],         mu           :             1[YES]
isconvex      :             1[YES],         LengthSY     :             4[YES]
>
```

Observe that the selected line search options are reported in the output. The full list of available line search options can be found in (Huang *et al.* 2016). At this time, a user-defined line search algorithm (`LineSearch_LS = 5`) cannot be specified via R, but support will be considered for a future release of **ManifoldOptim**.

4. Examples with Statistical Methods

This section demonstrates the use of **ManifoldOptim** in the three main methodologies presented in section 2: MADE, PFC, and envelope regression.

4.1. Minimum Average Deviance Estimation

We consider an application of MADE, which was summarized in section 2.1, on a fishing dataset. The data are adapted from Bailey *et al.* (2009), who were interested in how certain deep-sea fish populations were impacted when commercial fishing began in locations with deeper water than in previous years. The dataset has 147 observations and three continuous predictors: `density`, `log(meandepth)`, and `log(sweptarea)` which are, respectively, foliage density index, the natural logarithm of mean water depth per site, and the natural logarithm of the adjusted area of the site. The response `totabund` is the total number of fish counted per site. Our goal is to find a sufficient reduction of the three predictors to capture all regression information between the response and the predictors. To apply MADE to the fishing data, we assume a Poisson family for (2).

A full MADE program is included in the supplemental material. Step 2(b) of the iterative algorithm from section 2.1 is a Stiefel manifold optimization and can be carried out with **ManifoldOptim**. We make use of the `RCG` solver along with code for the objective function and its derivative with respect to B; these codes are based on the overall MADE objective (4).

The observed response is plotted against the fitted MADE response with $d = 1$ in Figure 1(a), and using the Poisson generalized linear regression model in Figure 1(b). Ideally, these points would follow a 0-intercept unit-slope line (plotted as the dashed line). However, the observation with $y = 1230$ seems to be an outlier under the estimated model.
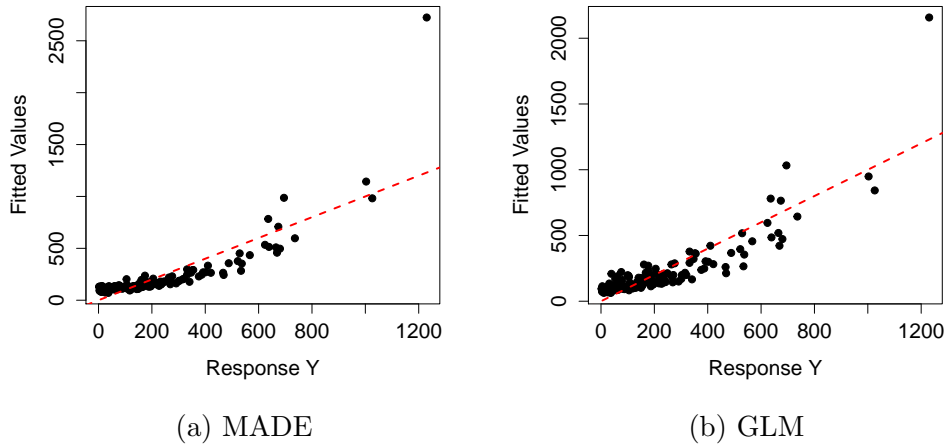

| (a) MADE | (b) GLM |

Figure 1: Plots of the fitted response $\hat{Y}$ against the observed Y.

4.2. Principal Fitted Components

We now consider a simulation study using the PFC model from section 2.2 to evaluate the performance of **ManifoldOptim**. The datasets were generated using $n = 300$ observations with

response $\mathbb{Y} \in \mathbb{R}^n$ from the normal distribution with mean 0 and variance 9. The matrix of basis function $\mathbb{F} \in \mathbb{R}^{n \times 2}$ was obtained as $(\mathbb{Y}, |\mathbb{Y}|)$ that is column-wise centered to have sample mean 0. We obtained Γ as two eigenvectors of a $p \times p$ generated positive definite matrix. The data matrix of the predictors $\mathbb{X} \in \mathbb{R}^{n \times p}$ was obtained as $\mathbb{X} = \mathbb{F}\Gamma^T + \mathbb{E}$, with error term $\mathbb{E} \in \mathbb{R}^{n \times p}$ generated from multivariate normal with mean 0 and variance Δ.

Two structures were used for the covariance Δ. The first was an unstructured covariance matrix $\Delta_u = U^T U$ where U is a $p \times p$ matrix with entries from the standard normal distribution. The second was an envelope structure $\Delta_e = \Gamma \Omega \Gamma^T + \Gamma_0 \Omega_0 \Gamma_0$ with $\Omega \in \mathbb{R}^{2 \times 2}$ and $\Omega_0 \in \mathbb{R}^{(p-2) \times (p-2)}$. The value of Γ is obtained from the first two eigenvectors of the unstructured Δ. The remaining eigenvectors are used for the value of Γ_0. These two structures correspond to the two models discussed in section 2.2. The likelihood (7) corresponds to the unstructured Δ_u while the likelihood function (6) corresponds to the envelope structure Δ_e. The parameters are then (Γ, Δ) in the unstructured PFC case, and $(\Gamma, \Omega, \Omega_0)$ for the envelope PFC. We note that in the case of the envelope structure, Γ_0 is the orthogonal completion of Γ so that $\Gamma\Gamma^T + \Gamma_0\Gamma_0^T = I$.

Estimation methods for these two PFC models exist. Cook and Forzani (2009) provide closed-form solution for the estimation of Γ and Δ in the unstructured case. For the envelope PFC, Cook (2007) provided a maximum likelihood estimation over a Grassmann manifold for Γ while Ω and Ω_0 have a closed-form that depend on Γ and Γ_0. The goal here is to evaluate the optimization and compare its performance to the closed-form solutions. Moreover, a product manifold is used so that the estimation of (Γ, Δ) is done once instead of a typical alternating procedure.

To proceed, we coded the objective and gradients functions under both setups. For each dataset generated in the simulation, the closed-form solutions were obtained and compared to the solutions obtained via **ManifoldOptim**. We compare the true parameter Γ to the estimate $\widehat{\Gamma}$ using the subspace distance $\rho(\Gamma, \widehat{\Gamma}) = \|(I - \widehat{\Gamma}\widehat{\Gamma}^T)\Gamma\|$ suggested by Xia *et al.* (2002). The true and estimated covariance were compared using $d(\Delta, \widehat{\Delta}) = \text{Trace}\{(\Delta - \widehat{\Delta})^T(\Delta - \widehat{\Delta})\}$. One hundred replications of the simulation were used. The estimated mean distances $d(\Delta, \widehat{\Delta})$ and $\rho(\Gamma, \widehat{\Gamma})$ are reported in Table 3 with their standard errors in parentheses.

The results indicated that the optimization performs adequately. The estimation using the optimization was carried out using $\widehat{\Sigma}$ and $\widehat{\Sigma}_{\text{res}}$ as initial values for Δ; the results were similar. The full code is provided as a supplement to this article.

Algorithm	$d(\Delta, \widehat{\Delta})$	$\rho(\Gamma, \widehat{\Gamma})$
Classical	2.67 (0.125)	0.17 (0.004)
ManifoldOptim	2.68 (0.125)	0.17 (0.004)

Table 2: Comparison of classical estimation and manifold optimization using ManifoldOptim of unstructured-Δ PFC parameters.

4.3. Envelope Models

We consider a simulation in Cook and Zhang (2015) to compare the results using **ManifoldOptim** to those using the three algorithms in Cook and Zhang (2015). We generated 150 independent observations using $Y_i \mid X_i \sim \text{Bernoulli}(\text{logit}^{-1}(\beta^T X_i))$, with $\beta = (\beta_1, \beta_2)^T = (0.25, 0.25)^T$ and $X_i \sim N(0, \Sigma_X)$. We let $\text{span}(\beta)$ be the principal eigenvector of Σ_X with eigenvalue 10

Algorithm	$d(\Omega, \widehat{\Omega})$	$d(\Omega_0, \widehat{\Omega}_0)$	$\rho(\Gamma, \widehat{\Gamma})$
Classical	0.03 (0.003)	0.36 (0.008)	0.28 (0.005)
ManifoldOptim	0.03 (0.003)	0.35 (0.008)	0.28 (0.005)

Table 3: Comparison of classical estimation and manifold optimization using ManifoldOptim of envelope-Δ PFC parameters.

and let the other eigenvalue be 0.1. This example is compelling because of the colinearity during construction of β. Consequently, this causes poor estimates when using a standard generalized linear model (GLM).

Algorithm	β_1	β_2
Truth	0.25	0.25
GLM	-0.12	0.63
Cook's Algorithm 1 with ManifoldOptim	0.25	0.24
Product Manifold Optimization	0.25	0.24

Table 4: Comparison of estimators between truth, Cook's algorithm 1 and product manifold using ManifoldOptim.

Three algorithms are considered in this section, namely GLM, Algorithm 1 in Cook and Zhang (2015), and a product manifold optimization. Both Algorithm 1 and the product manifold are constructed using functionality in ManifoldOptim. In the case of Algorithm 1, a Grassmann optimization in performed along with Fisher scoring in an iterative fashion. During the product manifold optimization the Grassmann manifold is optimized for Γ; η and α are estimated over the Euclidean manifold. Numerical matrix derivatives are calculated with ManifoldOptim instead of using the analytic version as in Cook and Zhang (2015) since that only pertains to Grassmann manifold optimization. In all cases the LRBFGS (Huang *et al.* 2015b) algorithm is used for optimizing over manifolds with the objective function given by equation 3.7 in Cook and Zhang (2015). To avoid poor local maxima, the optimization was repeated several times from random starting points. Out of the many estimates at convergence, the one with the largest likelihood value is selected. The full code is provided as a supplement to this article.

5. Conclusions

We have presented **ManifoldOptim**, an R package for optimization on Riemannian Manifolds. **ManifoldOptim** fills a need for manifold optimization in R by making the functionality of **ROPTLIB** accessible to R users. The package so far deals with optimization on the Stiefel manifold, Grassmann manifold, unit sphere manifold, the manifold of positive definite matrices, the manifold of low rank matrices, the orthogonal group, and Euclidean space. We discussed basic usage of the package, and demonstrated coding of problems in plain R or with C++. Furthermore, we demonstrated a product space of manifolds to optimize over multiple variables where each is constrained to its own manifold. Optimization of manifold-constrained parameters is a feature of emerging statistical methodology in areas such as sufficient dimension reduction and envelope models. We have discussed several applications of such models, and demonstrated ways in which **ManifoldOptim** could practically be applied. We hope that

availability of this package will facilitate exploration of these methodologies by statisticians and the R community.

References

Absil PA, Baker CG, Gallivan KA (2007). "Trust-region methods on Riemannian manifolds." *Foundations of Computational Mathematics*, **7**(3), 303–330.

Absil PA, Mahony R, Sepulchre R (2004). *Optimization Algorithms on Matrix Manifolds*. Princeton University Press, Princeton, NJ.

Absil PA, Mahony R, Sepulchre R (2008). *Optimization algorithms on matrix manifolds*. Princeton University Press, Princeton, NJ.

Adragni KP, Raim AM, Al-Najjar E (2016). "Minimum Average Deviance Estimation for Sufficient Dimension Reduction." arXiv:1611.00400.

Adragni KP, Wu S (2010). "R package for Grassmann Manifold Optimization." *Journal of Statistical Software*. URL http://cran.r-project.org/web/packages/GrassmannOptim/.

Anderson E, Bai Z, Bischof C, Blackford S, Demmel J, Dongarra J, Du Croz J, Greenbaum A, Hammarling S, McKenney A, Sorensen D (1999). *LAPACK Users' Guide*. Third edition. Society for Industrial and Applied Mathematics, Philadelphia, PA. ISBN 0-89871-447-8 (paperback).

Bailey D, Collins M, Gordon J, Zuur A, Priede I (2009). "Long-term changes in deep-water fish populations in the northeast Atlantic: a deeper reaching effect of fisheries?" *Proceedings of the Royal Society of London B: Biological Sciences*, pp. rspb–2009.

Berry MW, Drmac Z, Jessup ER (1999). "Matrices, Vector Spaces, and Information Retrieval." *SIAM Review*, **41**(2), 335–362.

Boumal N, Mishra B, Absil PA, Sepulchre R (2014). "Manopt, a Matlab Toolbox for Optimization on Manifolds." *Journal of Machine Learning Research*, **15**, 1455–1459. URL http://www.manopt.org.

Chikuse Y (2003). *Statistics on Special Manifolds*. Lecture Notes in Statistics; 174. Springer-Verlag, New York City, NY.

Comon P, Golub GH (1990). "Tracking a Few Extreme Singular Values and Vectors in Signal Processing." *IEEE Proceedings*, **78**(8), 1327–1343.

Cook RD (1998). *Regression Graphics*. New York: Wiley.

Cook RD (2007). "Fisher Lecture: Dimension Reduction in Regression." *Statistical Science*, **22**(1), 1–26.

Cook RD, Forzani L (2008). "Covariance Reducing Models: An Alternative to Spectral Modelling of Covariance." *Biometrika*, **95**(4), 799–812.

Cook RD, Forzani L (2009). "Likelihood-based Sufficient Dimension Reduction." *Journal of the American Statistical Association*, **104**(485), 197–208.

Cook RD, Li B, Chiaromonte F (2010). "Envelope Models for Parsimonious and Efficient Multivariate Linear Regression (with discussion)." *Statistica Sinica*, **20**(3), 927–1010.

Cook RD, Li L (2009). "Dimension Reduction in Regression with Exponential Family Predictors." *Journal of Computational and Graphical Statistics*, **18**(3), 774–791.

Cook RD, Su Z (2013). "Scaled envelopes: scale-invariant and efficient estimation in multivariate linear regression." *Biometrika*, **100**(4), 939–954.

Cook RD, Zhang X (2015). "Foundations for Envelope Models and Methods." *Journal of the American Statistical Association*, **110**(510), 599–611. doi:10.1080/01621459.2014.983235. http://dx.doi.org/10.1080/01621459.2014.983235, URL http://dx.doi.org/10.1080/01621459.2014.983235.

Eddelbuettel D, Francois R (2011). "Rcpp: Seamless R and C++ Integration." *Journal of Statistical Software*, **40**(1), 1–18. doi:10.18637/jss.v040.i08. URL https://www.jstatsoft.org/index.php/jss/article/view/v040i08.

Eddelbuettel D, François R (2016). *Exposing C++ functions and classes with Rcpp modules*. Rcpp version 0.12.3 as of January 10, 2016.

Eddelbuettel D, Sanderson C (2014). "RcppArmadillo: Accelerating R with high-performance C++ linear algebra." *Computational Statistics and Data Analysis*, **71**, 1054–1063. URL http://dx.doi.org/10.1016/j.csda.2013.02.005.

Edelman A, Arias T, Smith S (1998). "The Geometry of Algorithms with Orthogonality Constraints." *SIAM Journal of Matrix Analysis and Applications*, **20**(2), 303–353.

Fletcher R (1987). *Practical Methods of Optimization*. second edition. Wiley, Hoboken, NJ.

Huang W, Absil PA, Gallivan KA (2015a). "A Riemannian symmetric rank-one trust-region method." *Mathematical Programming*, **150**(2), 179–216.

Huang W, Absil PA, Gallivan KA, Hand P (2016). "ROPTLIB: an object-oriented C++ library for optimization on Riemannian manifolds." *Technical Report FSU16-14*, Florida State University.

Huang W, Gallivan KA, Absil PA (2015b). "A Broyden Class of Quasi-Newton Methods for Riemannian Optimization." *SIAM Journal on Optimization*, **25**(3), 1660–1685.

Lee J (2000). *Introduction to Topological Manifolds*. Second ed. edition. Springer, New York.

Lee J (2003). *Introduction to Smooth Manifolds*. Springer, New York.

Lippert R (2000). URL http://www-math.mit.edu/~lippert/sgmin.html.

Nocedal J, Wright S (1999). *Numerical Optimization*. Springer-Verlag.

Patel RV, Laub AJ, Van Dooren PM (1994). *Numerical Linear Algebra Techniques for Systems and Control*. IEEE Press, Piscataway, NJ.

R Core Team (2016). *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria. URL http://www.R-project.org/.

Ring W, Wirth B (2012). "Optimization methods on Riemannian manifolds and their application to shape space." *SIAM Journal on Optimization*, **22**(2), 596–627.

Sato H, Iwai T (2013). "A Riemannian optimization approach to the matrix singular value decomposition." *SIAM Journal on Optimization*, **23**(1), 188–212.

Su Z, Cook RD (2011). "Partial envelopes for efficient estimation in multivariate linear regression." *Biometrika*, p. asq063.

Su Z, Cook RD (2012). "Inner envelopes: efficient estimation in multivariate linear regression." *Biometrika*, **99**(3), 687–702.

Tu LW (2011). *Introduction to Manifolds.* 2nd edition. Springer-Verlag, New York.

Wong YC (1967). "Differential Geometry of Grassmann Manifolds." In *Proceedings of the National Academy of Science*, volume 57, pp. 589–594. USA.

Xia Y, Tong H, Li WK, Zhu LX (2002). "An adaptive estimation of dimension reduction space." *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **64**(3), 363–410.

Affiliation:

Kofi Placid Adragni
Department of Mathematics & Statistics
University of Maryland, Baltimore County
1000 Hilltop Circle
Baltimore, MD 21250
E-mail: kofi@umbc.edu
Phone: +1/410/455-2406
Fax: +1/410/455-1066
URL: http://www.umbc.edu/~kofi/