Journal of Statistical Software

April 2020, Volume 93, Issue 1. *doi: 10.18637/jss.v093.i01*

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

Sean R. Martin
University of Maryland, Baltimore County

Andrew M. Raim
U.S. Census Bureau

Wen Huang
Xiamen University

Kofi P. Adragni
University of Maryland, Baltimore County

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

Optimization on Riemannian manifolds has been a topic of much interest over the past years due to many important applications in computer vision, signal processing, numerical linear algebra, statistics, among others. However, the substantial background required to successfully design and apply Riemannian optimization algorithms is an impediment for many potential users. Ready-to-use tools are being developed to improve accessibility to Riemannian optimization. This article presents **ManifoldOptim** (Adragni, Martin, Raim, and Huang 2020), an R (R Core Team 2019) package for the optimization of real-valued functions on Riemannian manifolds. **ManifoldOptim** is a wrapper for **ROPTLIB** (Huang, Absil, Gallivan, and Hand 2017), a C++ library that provides a variety of cutting edge algorithms for optimization on manifolds and a framework for constructing such optimization problems.[1] **ROPTLIB** provides reliable and efficient implementations of many familiar optimization algorithms that have been extended to operate on Riemannian manifolds. The third author of this article (Wen Huang) is the creator of **ROPTLIB**, while the remaining authors were involved in the development and application of the R interface.

Consider the problem of minimizing $f(\mathrm{U})$ over $\mathrm{U} \in \mathcal{M}$ where $\mathcal{M}$ is a Riemannian manifold. 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. Stating a precise definition of a manifold requires mathematical exposition which is beyond this article; Lee (2000, 2003) and Tu (2011) provide introductions. Intuitively, some constrained spaces which are frequently encountered in practice are examples of manifolds. 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 positive definite matrices. We also consider spaces formed by taking Cartesian products of these manifolds.

Some optimization problems are readily solved without consideration of manifolds. Methods are abound in the literature for optimization problems without constraints or with linear constraints (Fletcher 1987; Nocedal and Wright 1999). Problems with nonlinear constraints sometimes become unconstrained by an appropriate transformation. For example, suppose $\mathcal{M} = \{\mathrm{U} \in \mathbb{R}^2 : \mathrm{U}_1^2 + \mathrm{U}_2^2 = 1\}$ is the unit circle in $\mathbb{R}^2$. Using the transformation

$$h(x) = \left(\cos\left(\frac{2\pi}{1+e^{-x}}\right), \sin\left(\frac{2\pi}{1+e^{-x}}\right)\right) : \mathbb{R} \to \mathcal{M},$$

minimization of $f(\mathrm{U})$ over $\mathrm{U} \in \mathcal{M}$ may be carried out practically without constraints by minimizing $f(h(x))$ over $x \in \mathbb{R}$.

Some optimization problems are not easily solved with classical methods, but constraints are naturally honored by restricting to an appropriate manifold. For example, let U be a $p \times d$ semi-orthogonal matrix where $\mathrm{U}^\top \mathrm{U} = \mathrm{I}_d$ 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 $f(\mathrm{U}) = f(\mathrm{UO})$ 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. Manifold optimization problems are often characterized by symmetry or invariance properties of the objective function.

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.

There is a growing literature on manifolds and manifold optimization. Optimization on Riemannian manifolds generalizes Euclidean optimization algorithms to curved and smoothed surfaces (Absil, Mahony, and Sepulchre 2008; Wong 1967; Edelman, Arias, and Smith 1998; Chikuse 2003). Applications are found in many areas including signal processing (Comon

[1]Currently, the **ROPTLIB** source code is included directly in the **ManifoldOptim** package code under the subdirectories: `src/Manifolds`, `src/Others`, `src/Problems`, `src/Solvers`. As of this writing, **ManifoldOptim** uses version 0.3 of **ROPTLIB**.

and Golub 1990), data mining (Berry, Drmac, and Jessup 1999), and control theory (Patel, Laub, and Van Dooren 1994). Many eigenvalue problems are, in fact, optimization problems on manifolds (Absil *et al.* 2008; Edelman *et al.* 1998). Manifold optimization appears in a wide variety of computational problems in the applied sciences, and has recently gained prominence in the statistics literature. Optimization over a Grassmann manifold is featured in several models for sufficient dimension reduction (SDR) in regression (Cook 1998). SDR methods include principal fitted components (Cook 2007), covariance reducing models (Cook and Forzani 2008), and likelihood-based sufficient dimension reduction (Cook and Forzani 2008; Cook and Li 2009; Cook and Forzani 2009). Adragni, Cook, and Wu (2012) provide some relevant examples. Recent methodology on envelopes for regression (Cook, Li, and Chiaromonte 2010; Su and Cook 2011, 2012; Cook and Su 2013; Cook and Zhang 2015) also relies on optimization over manifolds.

ManifoldOptim makes many of the optimization algorithms in the **ROPTLIB** library accessible to R users. Available algorithms include: the Riemannian trust-region Newton (Absil, Baker, and Gallivan 2007), Riemannian symmetric rank-one trust-region (RTRSR1) (Huang, Absil, and Gallivan 2015a), limited-memory RTRSR1 (Huang *et al.* 2015a), Riemannian trust-region steepest descent (Absil *et al.* 2008), Riemannian line-search Newton (Absil *et al.* 2008), Riemannian Broyden family (Huang, Gallivan, and Absil 2015b), Riemannian Broyden-Fletcher-Goldfarb-Shannon (RBFGS) (Ring and Wirth 2012; Huang *et al.* 2015b), limited memory RBFGS (Huang *et al.* 2015b), Riemannian conjugate gradients (Nocedal and Wright 1999; Absil *et al.* 2008; Sato and Iwai 2013), and Riemannian steepest descent (Absil *et al.* 2008).

In a typical use of **ManifoldOptim**, the user provides an objective function, its gradient and action of the Hessian, 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 (The MathWorks Inc. 2019) through its included `mex`-based wrapper. Interfaces to **ROPTLIB** for other high level languages, such as Julia (Bezanson, Karpinski, Shah, and Edelman 2012), have also been developed (Huang *et al.* 2016). Other manifold optimization software packages are found in the literature: **sg_min** written by Lippert (2008) which was adapted from Edelman *et al.* (1998), and **Manopt** of Boumal, Mishra, Absil, and Sepulchre (2014) are available in MATLAB. The **Pymanopt** package for Python (Van Rossum *et al.* 2019), introduced in Townsend, Koep, and Weichwald (2016), offers similar functionality as **Manopt**. In R, Adragni *et al.* (2012) developed **GrassmannOptim** specifically for Grassmann manifold optimization via a simulated annealing stochastic gradient algorithm. Material for a package called **rOptManifold** exists on the internet (see He, He, Huang, and Xie 2014); it appears to be a general framework for optimization over matrix manifolds in R, much like **ROPTLIB** with **ManifoldOptim**, which was not released for widespread use. To our knowledge, there are no other publicly available R packages for manifold optimization.

The following notations are adopted throughout this article: $\mathcal{G}_{d,p}$ 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 introduces several applications of manifold optimization which will be used to illustrate **ManifoldOptim**. Section 3 demonstrates basic usage of the package through brief examples. Section 4 presents more involved examples based on statistical methods introduced in Section 2. Discussions and conclusions follow in Section 5. The **ManifoldOptim** package is available from the Comprehensive R Archive Network (CRAN) at `https://CRAN.R-project.org/package=ManifoldOptim`. R code for the examples demonstrated in this article has been provided as supplementary material; see the included `README.txt` file for a description of the contents. The supplementary code has been tested with **ManifoldOptim** version 0.1.4, and results shown in this article have been produced with this version.

2. Optimization on manifolds

Consider the Brockett problem (Absil *et al.* 2008, Section 4), which amounts to minimizing the objective function

$$f(X) = \mathrm{Trace}(X^\top BXD), \quad X \in \mathbb{R}^{p\times d} \tag{1}$$

subject to $X^\top X = I_p$. Here, $D = \mathrm{Diag}(\mu_1, \ldots, \mu_p)$ 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 $\lambda_{p-d+1}, \ldots, \lambda_p$. This is essentially an eigenvalue problem reminiscent of the generalized Rayleigh quotient for discriminant analysis (Adragni *et al.* 2012). It will be later used in Section 3 to illustrate practical use of the package.

In statistical applications, optimization is primarily focused on maximizing a likelihood function to find maximum likelihood estimators (MLE). With multiple parameters constrained to different spaces, a common approach to maximum likelihood estimation is to optimize one parameter at a time while keeping the others fixed, alternating through such steps until meeting convergence criteria for the overall problem. This method of optimization, known as an alternating procedure, can provably converge to a global maximum only under specific conditions (Csiszar and Tusnady 1984). Alternating procedures may also involve different methods of optimization. Algorithm 1 of Cook and Zhang (2015) alternates between constrained optimization of variable estimates on the Grassmann manifold, and then optimization of variable estimates in Euclidean space using Fisher scoring iterations. To simplify these situations, **ManifoldOptim** supports a product space of manifolds which are composed from simpler manifolds and can include Euclidean spaces. Here, optimization algorithms work over all parameters jointly rather than alternating through partial optimizations. In our experience, joint optimization has produced comparable results to the alternating approach with a reduced programming burden. This is demonstrated in Sections 4.2 and 4.3.

We next present three algorithms in statistics that can be formulated as manifold optimization problems. The first is the recently developed minimum average deviance estimation method for SDR (Adragni 2017). 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 2017). This approach was first proposed by Xia, Tong, Li, and Zhu (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 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 assumed to be free of X. The canonical parameter $\theta(X)$ possesses the main information that connects Y to X; it relates to the mean function $\mathsf{E}(Y \mid X)$ through a link function g so that $g(\mathsf{E}(Y \mid X)) = \theta(X)$. 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 $\theta(X) = \vartheta(B^\top X)$ for some function ϑ, then $X \in \mathbb{R}^p$ can be replaced by $B^\top X \in \mathbb{R}^d$ in the regression of Y on X, and B is the basis of a dimension reduction subspace. MADE simultaneously estimates the reduction B and the unknown regression function ϑ from the sample.

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

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

using the first-order Taylor expansion

$$\begin{aligned}\vartheta(B^\top X_i) &\approx \vartheta(B^\top X_0) + [\nabla\vartheta(B^\top X_0)]^\top (B^\top X_i - B^\top X_0)\\ &= \alpha_0 + \gamma_0^\top B^\top (X_i - X_0),\end{aligned}$$

taking $\alpha_0 = \vartheta(B^\top X_0)$ and $\gamma_0 = \nabla\vartheta(B^\top X_0)$. The subscript in α_0 and γ_0 emphasizes 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 $L_{X_0}(\alpha_0, \gamma_0, B)$. The objective in MADE is to maximize the sum of local likelihoods

$$\begin{aligned}Q(B) &= \sum_{j=1}^{n} \max_{(\alpha_j,\gamma_j)} L_{X_j}(\alpha_j, \gamma_j, B)\\ &= \sum_{j=1}^{n} \max_{(\alpha_j,\gamma_j)} \left[\sum_{i=1}^{n} w(X_i, X_j) \log f(Y_i \mid \alpha_j + \gamma_j^\top B^\top (X_i - X_j))\right].\end{aligned} \tag{3}$$

Here, the $\hat{\alpha}_j \in \mathbb{R}$ and $\hat{\gamma}_j \in \mathbb{R}^d$ which maximize the inner summation are considered to be functions of B. Once a solution $\hat{B}$ is obtained for (3), predictions for given $X_0 \in \mathbb{R}^p$ may be computed as $\hat{Y}_0 = g^{-1}(\hat{\alpha}_0)$, using

$$(\hat{\alpha}_0, \hat{\gamma}_0) = \underset{(\alpha_0,\gamma_0)}{\arg\max}\, L_{X_0}(\alpha_0, \gamma_0, \hat{B}). \tag{4}$$

The kernel weights are taken to be

$$w(X_i, X_0) = \frac{K_{\mathrm{H}}(X_i - X_0)}{\sum_{\ell=1}^{n} K_{\mathrm{H}}(X_\ell - X_0)}, \quad K_{\mathrm{H}}(\mathrm{u}) = |\mathrm{H}|^{-1/2} K(\mathrm{H}^{-1/2}\mathrm{u}). \tag{5}$$

Here, $K(u)$ denotes one of the usual multidimensional kernel density functions and the bandwidth H is a $p \times p$ symmetric and positive definite matrix; for example, see Duong (2007). We will use the Gaussian kernel $K(x) = (2\pi)^{-p/2} \exp(-x^\top x/2)$ with $\mathrm{H} = h^2 I$. We also consider refined weights $w(X_i, X_0) = w(B^\top X_i, B^\top X_0)$ which make use of the unknown B; we will explicitly write $w(B^\top X_i, B^\top X_0)$ hereafter.

For any orthogonal matrix $V \in \mathbb{R}^{d \times d}$, $\gamma^\top B^\top = \gamma^\top V V^\top B^\top$, which implies that γ and B are not uniquely determined but obtained up to an orthogonal transformation. Furthermore, refined weights based on our choice of Gaussian kernel depend on B only through $BB^\top = BVV^\top B^\top$. In this setting, the MADE objective function (3) is invariant to orthogonal transformation of B in the sense that $Q(B) = Q(BV)$. Therefore the parameter space of B is a Grassmann manifold $\mathcal{G}_{d,p}$.

When the outcome is assumed to be Gaussian, MADE coincides with minimum average variance estimation (MAVE) introduced by Xia *et al.* (2002), who suggest an algorithm based on quadratic programming to fit the model. Computation becomes more challenging when other distributions for the outcome are considered. In the initial formulation of MADE, Adragni (2017) used an iterative algorithm to maximize (3), solving $(\alpha_j, \gamma_j) \in \mathbb{R}^{d+1}$ for $j = 1, \ldots, n$, optimizing $B \in \mathcal{S}_{d,p}$, and updating the refined weights in successive steps. In the present article, we directly optimize the objective function (3) over $\mathcal{G}_{d,p}$.

Appropriate selection of the bandwidth h is crucial to obtain a reasonable model. Taking h too small leads to overfitting, where the model fits the observed data very well but generalizes poorly to new observations. On the other hand, taking h too large leads to oversmoothing, where important features of the data are not taken into account. Here we will summarize a cross-validation approach to select h. Further details are given in Section 4.1.

To carry out K-fold cross-validation with a prespecified K, randomly partition the n observations into subsets $\mathcal{S}_1, \ldots, \mathcal{S}_K$. For each $k = 1, \ldots, K$ and a particular candidate value of h, let $\hat{B}_{\mathrm{CV}(k)}$ be the solution to (3) using the training set $\{1, \ldots, n\} \setminus \mathcal{S}_k$. Predictions for the validation set $j \in \mathcal{S}_k$ are computed as $\hat{Y}_{\mathrm{CV},j} = g^{-1}(\hat{\alpha}_{\mathrm{CV},j})$, where $(\hat{\alpha}_{\mathrm{CV},j}, \hat{\gamma}_{\mathrm{CV},j})$ are obtained from (4) using $\hat{B} = \hat{B}_{\mathrm{CV}(k)}$. The maximized local likelihood is computed as

$$\hat{Q}_{\mathrm{CV},j} = L_{X_j}(\hat{\alpha}_{\mathrm{CV},j}, \hat{\gamma}_{\mathrm{CV},j}, \hat{B}_{\mathrm{CV}(k)}).$$

Candidate values of h may be compared using

- mean square prediction error $\mathrm{MSPE} = n^{-1} \sum_{j=1}^{n} (Y_j - \hat{Y}_{\mathrm{CV},j})^2$,
- mean absolute prediction error $\mathrm{MAPE} = n^{-1} \sum_{j=1}^{n} |Y_j - \hat{Y}_{\mathrm{CV},j}|$, and
- MADE objective value $\mathrm{OBJ} = \sum_{j=1}^{n} \hat{Q}_{\mathrm{CV},j}$

obtained from cross-validation.

Offsets are fixed intercept terms which are commonly used in count regression, often to focus the analysis on a rate of exposure rather than on the raw counts. For the example in Section 4.1, an offset will help to avoid numerical issues due to large counts. When an offset w_i is available for the ith observation, it is simply added to the local regression function as

$$\alpha_0 + \gamma_0^\top B^\top (X_i - X_0) + w_i$$

in all previous expressions. Also, predictions are adjusted with the offset as $\hat{Y}_j = g^{-1}(\hat{\alpha}_j + w_i)$.

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. Using the stochastic nature of the predictors, Cook (2007) proposed the model

$$\mathrm{X}_y = \mu + \Gamma\beta \mathrm{f}_y + \Delta^{1/2}\varepsilon. \tag{6}$$

Here, X_y denotes the conditional X given $Y = y$ and $\mathrm{f}_y \in \mathbb{R}^r$ is a user-selected function of the response which helps capture the dependency of X on Y. The other parameters are $\mu \in \mathbb{R}^p$, a semi-orthogonal matrix $\Gamma \in \mathbb{R}^{p\times d}$, 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.

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

Cook (2007) showed that a sufficient reduction of X is $\eta^\top\mathrm{X}$, where $\eta = \Delta^{-1}\Gamma$, so that X can be replaced by $\eta^\top\mathrm{X}$ without loss of information about the regression of Y on X. However, as $\eta^\top\mathrm{X}$ is a sufficient reduction, $\mathrm{O}^\top\eta^\top\mathrm{X}$ is also sufficient for any $d\times d$ orthogonal matrix O. 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 $\Delta = \mathrm{Diag}(\sigma_1^2, \ldots, \sigma_p^2)$, the heterogeneous structure $\Delta = \Gamma\Omega\Gamma^\top + \Gamma_0\Omega_0\Gamma_0^\top$, and the general unstructured case $\Delta > 0$. 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 maximum likelihood estimation in PFC under the heterogeneous and general unstructured forms of Δ. Assuming that a sample 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 $(\mathrm{X}_y - \bar{\mathrm{X}})^\top$. Let $\mathbb{F}$ denote the $n \times r$ matrix with rows $(\mathrm{f}_y - \bar{\mathrm{f}})^\top$ 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 $\widehat{\Sigma} = \mathbb{X}^\top\mathbb{X}/n$ and $\widehat{\Sigma}_{\mathrm{fit}} = \widehat{\mathbb{X}}^\top\widehat{\mathbb{X}}/n$. For the heterogeneous and the unstructured Δ, the log-likelihood functions are respectively

$$\begin{aligned}
\mathcal{L}(\Omega, \Omega_0, \Gamma) &= -\frac{n}{2}\log|\Omega| - \frac{n}{2}\log|\Omega_0| - \mathrm{Trace}\left\{\Gamma^\top\widehat{\Sigma}_{\mathrm{res}}\Gamma\Omega^{-1} - \Gamma_0^\top\widehat{\Sigma}\Gamma_0\Omega_0^{-1}\right\}, &(7)\\
\mathcal{L}(\Delta, \Gamma) &= -\frac{n}{2}\log|\Delta| - \frac{n}{2}\mathrm{Trace}\left\{\left[\widehat{\Sigma} - \widehat{\Sigma}_{\mathrm{fit}}\Delta^{-1}\Gamma(\Gamma^\top\Delta^{-1}\Gamma)^{-1}\Gamma^\top\right]\Delta^{-1}\right\}. &(8)
\end{aligned}$$

These functions are real-valued, with parameter spaces expressed as Cartesian products of manifolds $(\Omega, \Omega_0, \Gamma) \in \mathcal{S}_d^+ \times \mathcal{S}_{p-d}^+ \times \mathcal{G}_{d,p}$ and $(\Delta, \Gamma) \in \mathcal{S}_p^+ \times \mathcal{G}_{d,p}$, respectively. Maximum likelihood estimators are obtained as

$$\begin{aligned}
(\widehat{\Omega}, \widehat{\Omega}_0, \widehat{\mathcal{S}}_\Gamma) &= \underset{(\Omega,\Omega_0,\mathcal{S}_\Gamma)}{\arg\max}\,\mathcal{L}(\Omega, \Omega_0, \Gamma), &(9)\\
(\widehat{\Delta}, \widehat{\mathcal{S}}_\Gamma) &= \underset{(\Delta,\mathcal{S}_\Gamma)}{\arg\max}\,\mathcal{L}(\Delta, \Gamma). &(10)
\end{aligned}$$

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

$$Y = \mu + \beta X + \varepsilon, \tag{11}$$

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 $\mathcal{S} \subseteq \mathbb{R}^r$ so that

$$\mathrm{Q}_{\mathcal{S}} Y \mid X \sim \mathrm{Q}_{\mathcal{S}} Y \quad \text{and} \quad \mathrm{Q}_{\mathcal{S}} Y \perp\!\!\!\perp \mathrm{P}_{\mathcal{S}} Y \mid X, \tag{12}$$

where $\sim$ means identically distributed, $\mathrm{P}_{\mathcal{S}}$ projects onto the subspace $\mathcal{S}$ and $\mathrm{Q}_{\mathcal{S}} = I_r - \mathrm{P}_{\mathcal{S}}$. For any $\mathcal{S}$ with those properties, $\mathrm{P}_{\mathcal{S}} Y$ carries all of the material information and perhaps some immaterial information, while $\mathrm{Q}_{\mathcal{S}}$ carries just immaterial information. Denoting $\mathcal{B} := \operatorname{span}(\beta)$, expressions (12) hold if and only if $\mathcal{B} \subseteq \mathcal{S}$ and $\Sigma = \Sigma_{\mathcal{S}} + \Sigma_{\mathcal{S}^\perp}$, where $\Sigma_{\mathcal{S}} = \mathsf{VAR}(\mathrm{P}_{\mathcal{S}} Y)$ and $\Sigma_{\mathcal{S}^\perp} = \mathsf{VAR}(\mathrm{Q}_{\mathcal{S}} Y)$. 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 (11), 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 (GLMs). The response belongs to an exponential family of the form (2). Let $\mathcal{C}(\vartheta) = y\vartheta - b(\vartheta)$ and $W(\vartheta) = \mathcal{C}''(\vartheta)/\mathsf{E}(\mathcal{C}''(\vartheta))$. They reparameterized $\vartheta(\alpha, \beta) = \alpha + \beta^\top X$ to $\vartheta(\alpha, \beta) = a + \beta^\top\{X - \mathsf{E}(WX)\}$ with $a = \alpha + \beta^\top \mathsf{E}(WX)$ so that a and β are orthogonal. The asymptotic variance of $\hat{\beta}$ is then obtained as $\mathsf{AVAR}(\sqrt{n}\hat{\beta}) = \mathbf{V}_{\beta\beta}(a, \beta) = \{\mathsf{E}(-\mathcal{C}''\mathsf{E}\{[X - \mathsf{E}(WX)][X - \mathsf{E}(WX)]^\top\}\}$. With a reparameterization $\beta = \Gamma\eta$, the parameter η is estimated using a Fisher scoring method fitting the GLM of Y on $\Gamma^\top X$. The partially maximized log-likelihood for Γ is

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

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

3. Using ManifoldOptim

We now describe use of the **ManifoldOptim** package. The primary function for users 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())
```

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`.

A problem encapsulates the objective function to be minimized, and, optionally, the corresponding gradient and Hessian functions. Analytical expressions of the gradient and Hessian are desirable, but may be either tedious or intractable to compute. Numerical approximations to the gradient and Hessian via finite differences are used by default. There are two options for constructing a problem with **ManifoldOptim**: writing the problem in R as an '`RProblem`' and writing the problem in C++ as a '`ManifoldOptimProblem`'. **ManifoldOptim** treats the optimization variable as a single vector which the user must reshape 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 2019).

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**.

The focus of the initial release **ManifoldOptim** has been on ease of use. **ROPTLIB** provides additional constructs to assist C++ programmers in avoiding redundant computations and redundant memory usage, which can significantly improve run time during optimization (Huang *et al.* 2016). This functionality is currently not yet exposed in **ManifoldOptim**, but may be considered in future versions.

Note that optimization outputs may differ slightly from those reported in the following examples depending on the computer used to run them. This is in part because solutions on Stiefel/Grassman manifolds are not necessarily unique. Additionally, the optimization algorithms are numerical in nature and may differ based on the computer used. Such differences are common in numerical algorithms and are the subject of work to better quantify how well numerical algorithms run on different computer architectures (Louboutin, Lange, Herrmann, Kukreja, and Gorman 2017).

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

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

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 RTRSR1 (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)

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

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 and ensure that `B` is symmetric.

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

The objective and gradient functions are coded in R as follows.

```
R> tx <- function(x) {
+    matrix(x, n, p)
+  }
R> f <- function(x) {
+    X <- tx(x)
+    Trace(t(X) %*% B %*% X %*% D)
+  }
R> 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-provided 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’.

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

The ‘Module’ construct (Eddelbuettel and François 2016) has been utilized in the user interface of **ManifoldOptim**. When constructing a module in R, the user implicitly creates a C++ problem object which can be accessed by the C++ solver, eliminating one source of potential overhead.

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.

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

The argument IsCheckParams = TRUE requests some 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.

```
R> 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.

```
R> 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.

```
R> 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"    "message"
```

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

```
R> head(tx(round(res$xopt, digits = 4)))

        [,1]    [,2]    [,3]    [,4]    [,5]
[1,]  0.1688 -0.1100 -0.0876 -0.0396 -0.0520
[2,] -0.0075  0.1275 -0.1243  0.0486  0.1053
[3,]  0.0956 -0.0249  0.0396 -0.0008 -0.0134
[4,] -0.0381 -0.1710 -0.0477 -0.0142  0.0660
[5,] -0.1239 -0.0207  0.1199 -0.1236 -0.1112
[6,] -0.0296 -0.0049  0.0511 -0.1352 -0.0129
```

We may now compare the solution from the optimizer to the closed-form solution.

```
R> eig <- eigen(B)
R> X.star <- eig$vectors[, seq(n, n-p+1)]
R> f(res$xopt)

[1] -474.4818

R> f(as.numeric(X.star))

[1] -474.4819
```

The solutions have objective values which match to three decimal places; the precision from the optimizer can be further increased by specifying a smaller tolerance.

Solvers such as `"RNewton"` (Absil *et al.* 2008) make use of the Hessian function 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.* (2008). 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.

```
R> h <- function(x, eta) {
+    2 * (D %x% B) %*% eta
+  }
R> 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.

ROPTLIB provides a diagnostic to check correctness of the gradient and Hessian. It produces two outputs: the first uses the starting value, and the second uses the solution obtained from the optimizer. If the quantity `(fy - fx) / <gfx, eta>` is approximately 1 for some interval within the first output, it is an indication that the gradient function is correct. If the quantity `(fy - fx - <gfx, eta>) / <0.5 eta, Hessian eta>` is approximately 1 for some interval within the second output, it indicates that the Hessian function is correct. See the **ROPTLIB** manual for details. The diagnostic is requested by setting the `IsCheckGradHess` option in the solver.

```
R> solver.params <- get.solver.params(IsCheckGradHess = TRUE)
R> res <- manifold.optim(prob, mani.defn, method = "RNewton",
+    mani.params = mani.params, solver.params = solver.params, x0 = x0)
```

The first output validates the gradient function.

```
i:0,|eta|:1.000e+02,(fy-fx)/<gfx,eta>:-1.463e-04,(fy-fx-<gfx,eta>)/<0.5 eta,
  Hessian eta>:-2.524e+01
...
i:28,|eta|:3.725e-07,(fy-fx)/<gfx,eta>:1.000e+00,(fy-fx-<gfx,eta>)/<0.5 eta,
  Hessian eta>:-4.147e-01
i:29,|eta|:1.863e-07,(fy-fx)/<gfx,eta>:1.000e+00,(fy-fx-<gfx,eta>)/<0.5 eta,
  Hessian eta>:2.441e-01
i:30,|eta|:9.313e-08,(fy-fx)/<gfx,eta>:1.000e+00,(fy-fx-<gfx,eta>)/<0.5 eta,
  Hessian eta>:-1.669e+01
...
i:34,|eta|:5.821e-09,(fy-fx)/<gfx,eta>:1.000e+00,(fy-fx-<gfx,eta>)/<0.5 eta,
  Hessian eta>:5.574e+02
```

The second output validates the Hessian function.

```
i:0,|eta|:1.000e+02,(fy-fx)/<gfx,eta>:1.556e+03,(fy-fx-<gfx,eta>)/<0.5 eta,
  Hessian eta>:3.823e-04
...
i:21,|eta|:4.768e-05,(fy-fx)/<gfx,eta>:2.939e+00,(fy-fx-<gfx,eta>)/<0.5 eta,
  Hessian eta>:1.000e+00
i:22,|eta|:2.384e-05,(fy-fx)/<gfx,eta>:1.970e+00,(fy-fx-<gfx,eta>)/<0.5 eta,
  Hessian eta>:1.000e+00
i:23,|eta|:1.192e-05,(fy-fx)/<gfx,eta>:1.485e+00,(fy-fx-<gfx,eta>)/<0.5 eta,
  Hessian eta>:1.000e+00
...
i:34,|eta|:5.821e-09,(fy-fx)/<gfx,eta>:1.008e+00,(fy-fx-<gfx,eta>)/<0.5 eta,
  Hessian eta>:3.368e+01
```

3.2. Coding a problem in C++

For optimization problems which require intensive computation or which will be evaluated repeatedly, it may be worth investing some additional effort to write the problem in C++. A working knowledge of C++ programming and **Rcpp** will be assumed here; R users lacking this background can refer to Eddelbuettel (2013) as a starting point. We now illustrate the Brockett problem in C++. To proceed, we will consider the '`ManifoldOptimProblem`' class within **ManifoldOptim**. A '`ManifoldOptimProblem`' object has objective, gradient, and Hessian functions coded in C++ using the **Armadillo** library (Sanderson and Curtin 2016).[2] The user implements this problem type by extending the abstract '`ManifoldOptimProblem`' class.

The '`BrockettProblem`' class can be written as follows.

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

using namespace Rcpp;

class BrockettProblem : public ManifoldOptimProblem
{
public:
    BrockettProblem(const arma::mat& B, const arma::mat& D)
    : ManifoldOptimProblem(), _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::vec gradFun(const arma::vec& x) const {
        arma::mat X;
        tx(X, x);
        return reshape(2 * _B * X * _D, x.n_elem, 1);
    }
    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);
    }
```

[2] To make the distinction between **Armadillo** and **RcppArmadillo** clear, **Armadillo** is a C++ library for linear algebra, and **RcppArmadillo** enables use of **Armadillo** from **Rcpp** programs.

```
    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<arma::mat,arma::mat>()
    .method("objFun", &BrockettProblem::objFun)
    .method("gradFun", &BrockettProblem::gradFun)
    .method("hessEtaFun", &BrockettProblem::hessEtaFun)
    .method("GetB", &BrockettProblem::GetB)
    .method("GetD", &BrockettProblem::GetD)
    ;
}')
```

Some points to mention about the C++ code are:

1. `ManifoldOptim` has been specified in the `Rcpp::depends` attribute. This ensures that the **ManifoldOptim** C++ library is utilized during compilation and linking.

2. The constructor `BrockettProblem(const arma::mat& B, const arma::mat& D)` creates new instances of a '`BrockettProblem`' for given B and D matrices. It begins by initializing its '`ManifoldOptimProblem`' base class.

3. We defined an empty destructor `~BrockettProblem()`; no action is necessary here because we did not dynamically allocate memory.

4. The `tx` function is responsible for reshaping the flat vector `x` into variable(s) specific to the problem. In this case, we simply rearrange the vector `x` into an `n` by `p` matrix `X` using the **Armadillo** `reshape` function. Note that `X` is passed by reference, so that the value set by the `tx` function is available to the caller. If `x` were to contain multiple variables, the `tx` function could be extended to have multiple output arguments.

5. The objective, gradient, and Hessian functions make use of `tx` before carrying out the computations described in Section 3.1. If no gradient or Hessian functions are specified here, the `gradFun` and `hessEtaFun` functions in the '`ManifoldOptimProblem`' base class will be used to provide numerical derivatives.

6. The return type of `gradFun` is a vector, but our formulation of the Brockett gradient function in Section 3.1 produces a matrix. Therefore, we explicitly reshape the matrix into a column vector via the **Armadillo** `reshape` function.

7. In addition to defining the '`BrockettProblem`' class, we define 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.

8. If `gradFun` and `hessEtaFun` are left at their numerical defaults, but it is desired to call them from R, they may be defined as follows for inclusion in the module.

   ```
   arma::vec gradFun(const arma::vec& x) const
   {
       return ManifoldOptimProblem::gradFun(x);
   }

   arma::vec hessEtaFun(const arma::vec& x,
       const arma::vec& eta) const
   {
       return ManifoldOptimProblem::hessEtaFun(x, eta);
   }
   ```

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 details.

Using the same B and D as in the previous section, we can now invoke the constructor through R.

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

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

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

We use the same initial value as before. Our **Rcpp** module allows us to call the problem functions from R, which makes them easy to test.

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

Note that, because `x0` is taken to be a flat vector of length `n * p`, `gradFun` and `hessEtaFun` both return a vector of length `n * p`. We may now invoke the solver.

```
R> 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 be to reshape it from a vector to a matrix.

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

The result is close to the one obtained in the previous section and is therefore not shown.

3.3. 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

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

To code this problem, let us first generate an example dataset. We make use of the **mvtnorm** package (Genz *et al.* 2017) for multivariate normal calculations.

```
R> set.seed(1234)
R> n <- 400
R> p <- 3
R> mu.true <- rep(1/sqrt(p), p)
R> Sigma.true <- diag(2, p) + 0.1
R> y <- 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.

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

We have constructed 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. The `dmvnorm` function requires the `sigma` argument to be a symmetric matrix, but the `Sigma` provided by the optimizer may not be exactly symmetric due to numerical error, so we symmetrize it to ensure that the optimization can proceed.

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.

```
R> mani.defn <- get.product.defn(get.sphere.defn(p), get.spd.defn(p))
R> mani.params <- get.manifold.params()
R> solver.params <- get.solver.params(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.

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

The following result is produced.

```
R> tx(res$xopt)
$mu
[1] 0.5076017 0.7054302 0.4946805

$Sigma
           [,1]      [,2]       [,3]
[1,]  2.0005530 0.1362844 -0.1718265
[2,]  0.1362844 2.0289096  0.2664935
[3,] -0.1718265 0.2664935  2.0936074
```

If exact symmetry is needed for Σ, the symmetric part can be extracted as the estimate.

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

To implement this problem in C++, consider the following '`ConstrainedMLEProblem`' class and suppose it is saved to a file named `constrained-mle.cpp`.

```
// [[Rcpp::depends(RcppArmadillo,ManifoldOptim)]]
#include <RcppArmadillo.h>
#include <ManifoldOptim.h>
using namespace Rcpp;

class ConstrainedMLEProblem : public ManifoldOptimProblem {
public:
    ConstrainedMLEProblem(const arma::mat& Y)
    : ManifoldOptimProblem(), _Y(Y) { }

    virtual ~ConstrainedMLEProblem() { }

    double objFun(const arma::vec& x) const {
        size_t n = _Y.n_rows; size_t p = _Y.n_cols;
        arma::vec mu(p); arma::mat Sigma(p,p);
        tx(mu, Sigma, x);

        double ll = -0.5 * n * p * log(2*arma::datum::pi);
        ll -= 0.5 * n * arma::log_det(Sigma).real();
        for (size_t i = 0; i < n; i++) {
            arma::vec q_i = trans(_Y.row(i)) - mu;
            ll -= 0.5 * arma::dot(q_i, arma::solve(Sigma, q_i));
        }
        return -ll;
    }
    void tx(arma::vec& mu, arma::mat& Sigma, const arma::vec& x) const {
        size_t p = _Y.n_cols;
        mu = x(arma::span(0, p-1));
        Sigma = x(arma::span(p, p*(p+1)-1));
        Sigma.reshape(p, p);
    }
    const arma::mat& GetY() const { return _Y; }
private:
    arma::mat _Y;
};

RCPP_MODULE(NormalMLE_module) {
    class_<ConstrainedMLEProblem>("ConstrainedMLEProblem")
    .constructor<arma::mat>()
    .method("objFun", &ConstrainedMLEProblem::objFun)
    .method("GetY", &ConstrainedMLEProblem::GetY)
    ;
}
```

Here we are computing the log-likelihood manually using **Armadillo**, and do not need to symmetrize the `Sigma` proposed by the optimizer as we did before. We now compile the code with **Rcpp** and create an instance of the problem with the same `y` generated previously.

```
R> sourceCpp("constrained-mle.cpp")
R> prob_cpp <- new(ConstrainedMLEProblem, y)
```

Replacing the '`RProblem`' with the '`ConstrainedMLEProblem`' and proceeding with the same optimization yields a similar result (and is therefore not shown).

```
R> res_cpp <- manifold.optim(prob_cpp, mani.defn, method = "LRBFGS",
+    mani.params = mani.params, solver.params = solver.params, x0 = x0)
R> tx(res$xopt)
```

3.4. Customizing line search

Line search algorithms are analogous to the method of gradient descent. As described in detail in Absil *et al.* (2008), they are based on an update

$$x_{k+1} = x_k + t_k \eta_k, \tag{13}$$

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 3.3. Several line search parameters are specified below. In particular, `LineSearch_LS = 1` corresponds to the Wolfe line search method.

```
R> solver.params <- get.solver.params(Tolerance = 1e-4, IsCheckParams = TRUE,
+    LineSearch_LS = 1, Accuracy = 2e-4, Initstepsize = 1/2)
```

Now, using the `"LRBFGS"` solver results in the following output.

```
R> 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  :            0.5[YES]
Minstepsize   :   2.22045e-16[YES],    Maxstepsize   :           1000[YES]
Accuracy      :        0.0002[YES],    Finalstepsize :              1[YES]
Num_pre_funs  :             0[YES],    InitSteptype  :     QUADINTMOD[YES]
LRBFGS METHOD PARAMETERS:
nu            :        0.0001[YES],    mu            :              1[YES]
isconvex      :             0[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 MADE, PFC, and envelope regression methodologies presented in Section 2.

4.1. Minimum average deviance estimation

We consider an application of MADE to the fishing dataset from the **COUNT** package (Hilbe 2016). The data are adapted from Bailey, Collins, Gordon, Zuur, and Priede (2009), who were interested in how certain deep-sea fish populations were impacted when commercial

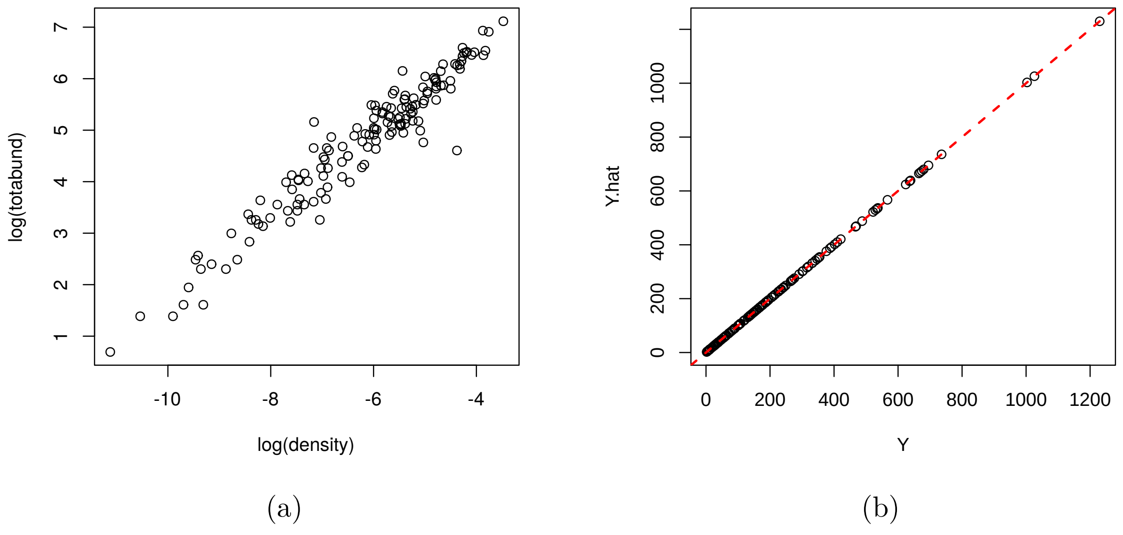

Figure 1: Exploratory results for the fishing dataset: (a) a scatterplot of `log(density)` against `log(totabund)`; (b) observed `totabund` values versus predictions from a Poisson GLM with `log(density)` as a covariate.

fishing began in locations with deeper water than in previous years. The dataset has 147 observations and three continuous predictors: `density`, `meandepth`, and `sweptarea`, which are, respectively, foliage density index, mean water depth per site, and the adjusted area of the site. The predictors are centered to have mean zero and scaled to have unit variance. The response `totabund` is the total number of fish counted per site. We use `log(sweptarea / 100)` as a fixed offset term.

```
R> data("fishing", package = "COUNT")
R> X <- model.matrix(~ density + meandepth + sweptarea - 1, data = fishing)
R> Xc <- scale(X, TRUE, TRUE)
R> y <- fishing$totabund
R> offs <- log(fishing$sweptarea / 100)
R> n <- nrow(Xc)
R> p <- ncol(Xc)
R> d <- 1
```

Our goal is to find a sufficient reduction of the three predictors to capture all regression information between the response and the predictors. Figure 1(a) reveals that the variable `log(density)` is highly correlated with `log(totabund)` in this dataset. Making use of this transformation, consider the following Poisson GLM.

```
R> glm.out <- glm(totabund ~ log(density) + meandepth + sweptarea +
+    offset(offs), family = poisson(), data = fishing)
R> y.hat <- predict(glm.out, type = "response")
```

(Note that uncentered and unscaled predictors were used here.) This leads to an almost perfect fit of the observed data shown in Figure 1(b). As an illustrative example of MADE methodology, we will learn the logarithmic nature of the relationship between `density` and `totabund` from the observed data. The dimension of the reduction $d = 1$ will be fixed throughout the analysis.

We now present an abbreviated version of the MADE code to explain how it is implemented using **ManifoldOptim**. The reader should refer to the supplemental materials for a fully composed MADE program. We begin by writing the MADE objective function (3) in R, focusing specifically on Poisson outcomes.

```
R> tx <- function(theta) {
+    matrix(theta, p, d)
+ }
R> f <- function(theta) {
+    B <- tx(theta)
+    objvals <- numeric(n)
+    for (j in 1:n) {
+      ww <- kern.weights(X %*% B, X[j, ] %*% B, h = h)
+      res <- solve.local(y, X, x0 = X[j, ], B = B, h = h,
+        offset = offs)
+      a.j <- res[1]; b.j <- res[-1]
+      X.j <- matrix(X[j, ], n, p, byrow = TRUE)
+      theta.j <- a.j + (X - X.j) %*% B %*% b.j + offs
+      logf <- dpois(y, exp(theta.j), log = TRUE)
+      objvals[j] <- sum(ww * logf)
+    }
+    return(-sum(objvals))
+ }
```

Here, `h` is a prespecified bandwidth, `kern.weights` is a function to compute kernel weights as in (5), and `solve.local` computes local regression coefficients by solving (4). We define an '`RProblem`' based on the objective function.

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

The gradient and Hessian are left unspecified so that numerical approximations will be used if needed. We now invoke the `manifold.optim` function using the `"RTRSD"` solver on a Grassmann manifold.

```
R> solver.params <- get.solver.params(Tolerance = 1e-4, Max_Iteration = 100)
R> mani.defn <- get.grassmann.defn(p, d)
R> res <- manifold.optim(prob, mani.defn, method = "RTRSD",
+    solver.params = solver.params, x0 = as.numeric(B.init))
```

A `made` function can now be defined to encapsulate the '`RProblem`' and the call to **ManifoldOptim**. We can also define a `predict` function to compute predictions via (4). A typical call to these functions would appear as follows.

```
R> made.out <- made(y = y, X = Xc, offset = offs, h = 1,
+    B.init = as.matrix(c(0, 1)))
R> pred.out <- predict(made.out, X.new = X[1:5, ], offset.new = offs[1:5])
```

h	MSPE	MAPE	OBJ
0.029487	1.584051e+07	4.267500e+02	−3867.20
0.040358	3.748522e+45	5.049767e+21	−3930.46
0.051229	8.410842e+22	2.391999e+10	−2718.32
0.062101	1.405407e+04	5.066000e+01	−2279.33
0.072972	1.160946e+04	3.783000e+01	−1940.06
0.083844	3.467290e+03	9.660000e+00	−572.26
0.094715	3.478510e+03	1.016000e+01	−590.50
0.105586	3.492600e+03	1.068000e+01	−610.02
0.116458	3.508750e+03	1.129000e+01	−630.26
0.127329	3.525610e+03	1.189000e+01	−650.61
0.138200	3.543140e+03	1.249000e+01	−670.64
0.149072	3.561170e+03	1.307000e+01	−690.10
0.159943	3.579530e+03	1.364000e+01	−708.93
0.170815	3.519300e+03	1.350000e+01	−727.16
0.181686	3.005800e+02	1.008000e+01	−744.89
0.192557	3.264500e+02	1.063000e+01	−762.25
0.203429	3.527200e+02	1.118000e+01	−779.30
0.214300	3.798100e+02	1.173000e+01	−796.19

Table 2: Quantities measuring goodness-of-fit in fishing cross-validation study of bandwidth.

With an implementation of the MADE algorithm, we now turn to the selection of an appropriate bandwidth h for the fishing data. We have taken $B = (0, 1, 0)^\top$ as the initial value for all optimizations. This starting value provides a test for our implementation; it emphasizes the variable `meandepth`, while our exploratory analysis suggests that the response depends primarily on `density` and therefore we expect the solution $\hat{B} \approx (1, 0, 0)^\top$. We performed 5-fold cross-validation as described in Section 2.1 based on an evenly spaced grid of 100 bandwidths,

$$\begin{aligned} h &= \left[0.08 + \frac{j}{100-1}(3.0 - 0.08)\right] n^{-1/(d+4)}, \quad j = 0, \ldots, 99, \\ &\approx (0.029487, 0.040358, \ldots, 1.105752). \end{aligned}$$

Here, $n^{-1/(d+4)}$ is the order of the asymptotically optimal bandwidth; see Adragni (2017) for discussion on this. Note that our MADE implementation encounters numerical issues if the bandwidth is taken smaller than the minimum value of the grid. Table 2 displays the MSPE, MAPE, and OBJ quantities obtained using the first 18 values of the grid. Recall that smaller values of MSPE and MAPE are desired, while larger values of OBJ are preferred. The best values of MAPE and OBJ are obtained with $h = 0.083844$. However, the MSPE drastically improves when h is increased to 0.181686; this indicates that at least one observation was being overfit with $h \leq 0.170815$. As h is increased beyond 0.214300, the MSPE, MAPE, and OBJ all gradually worsen (results are not shown).

Using the bandwidth $h = 0.181686$ selected by cross-validation, we obtain the fit

$$\hat{B}_{\text{made}} = \begin{pmatrix} -0.9989 \\ 0.0458 \\ 0.0049 \end{pmatrix}, \quad Q(\hat{B}_{\text{made}}) = -741.1762.$$

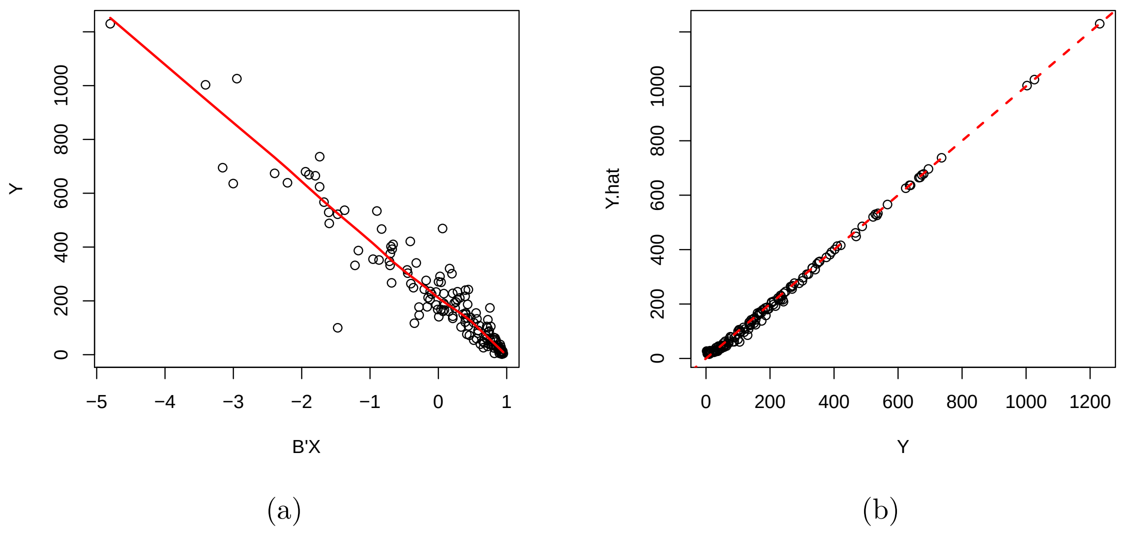

Figure 2: Fit of MADE to fishing dataset using h determined by cross-validation: (a) the estimated reduction $\hat{B}^\top X$ versus the response Y, with LOESS curve in red; (b) the observed response Y versus the predicted response $\hat{Y}$.

As anticipated, the reduction emphasizes the first coordinate corresponding to `density`, and deemphasizes the second and third coordinates corresponding to `meandepth` and `sweptarea`. Figure 2 shows that MADE has recovered the logarithmic nature of the relationship between `density` and the response `totabund`.

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 functions $\mathbb{F} \in \mathbb{R}^{n\times 2}$ was obtained as $(\mathbb{Y}, |\mathbb{Y}|)$ and 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 $\mathbb{X} = \mathbb{F}\Gamma^\top + \mathbb{E}$, with error term $\mathbb{E} \in \mathbb{R}^{n\times p}$ generated from the multivariate normal distribution with mean 0 and variance Δ.

Two structures were used for the covariance Δ. The first was an unstructured covariance matrix $\Delta_\mathrm{u} = U^\top U$ where U is a $p \times p$ matrix with entries from the standard normal distribution. The second was an envelope structure $\Delta_\mathrm{e} = \Gamma\Omega\Gamma^\top + \Gamma_0\Omega_0\Gamma_0^\top$ 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 (8) corresponds to the unstructured Δ_u while the likelihood function (7) 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 $\Gamma\Gamma^\top + \Gamma_0\Gamma_0^\top = I$.

Estimation methods for these two PFC models exist. Cook and Forzani (2009) provide closed-form solutions for 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 depends on Γ and Γ_0. Our implementation relies on three R packages: **MASS** (Venables and Ripley 2002), **ldr** (Adragni and Raim 2014), and **mvtnorm** (Genz *et al.* 2017). The objective function is coded as follows.

```
R> f <- function(x) {
+    Deltahat <- tx(x)
+    InvDeltahat <- solve(Deltahat)
+    Gam <- fit$Gammahat
+    temp0 <- -(n * p/2) * log(2 * pi)
+    temp1 <- -(n/2) * log(det(Deltahat))
+    temp2 <- -(n/2) * Trace((Sigmahat - Sfit %*% InvDeltahat %*% Gam %*%
+      solve(t(Gam) %*% InvDeltahat %*% Gam) %*% t(Gam)) %*% InvDeltahat)
+    loglik <- Re(temp0 + temp1 + temp2)
+    return(-loglik)
+  }
R> tx <- function(x) {
+    S <- matrix(x, p, p)
+    S[lower.tri(S)] <- t(S)[lower.tri(S)]
+    return(S)
+  }
```

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 that alternates between holding certain values constant and optimizing over them.

To proceed, we coded the objective function under both setups. For each dataset generated in the simulation, the closed-form solutions were obtained and compared to the solutions obtained via **ManifoldOptim**. A call for the optimization involves the following snippet.

```
R> mod <- Module("ManifoldOptim_module", PACKAGE = "ManifoldOptim")
R> prob <- new(mod$RProblem, f)
R> mani.params <- get.manifold.params(IsCheckParams = TRUE)
R> solver.params <- get.solver.params(DEBUG = 0, Tolerance = 1e-12,
+    Max_Iteration = 1000, IsCheckParams = TRUE, IsCheckGradHess = FALSE)
R> mani.defn <- get.spd.defn(p)
R> res <- manifold.optim(x0 = Sres, prob, method = "RTRSR1",
+    mani.defn = mani.defn, mani.params = mani.params,
+    solver.params = solver.params)
```

We compared the true parameter Γ to the estimate $\widehat{\Gamma}$ using the subspace distance $\rho(\Gamma, \widehat{\Gamma}) = \|(I - \widehat{\Gamma}\widehat{\Gamma}^\top)\Gamma\|$ suggested by Xia *et al.* (2002). The true and estimated covariance were compared using $d(\Delta, \widehat{\Delta}) = \mathrm{Trace}\{(\Delta - \widehat{\Delta})^\top(\Delta - \widehat{\Delta})\}$. One hundred replications were used for the simulation. The estimated mean distances $d(\Delta, \widehat{\Delta})$ and $\rho(\Gamma, \widehat{\Gamma})$ are reported in Tables 4 and 3 with their standard errors in parentheses.

The results indicate that both forms of optimization perform comparably. Thus, the product manifold in **ManifoldOptim** provides an alternative form of optimization which can ease implementation. Also note that the difference between the true and estimated covariance is

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

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

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

significantly less for the envelope-Δ PFC, since there are less parameters to estimate in this case. Optimization was carried out using $\widehat{\Sigma}$ and $\widehat{\Sigma}_{\text{res}}$ as initial values for Δ; the results were similar to the classical closed form method. The full code is provided as a supplement to this article.

4.3. Envelope models

We consider a simulation in Cook and Zhang (2015) to compare the results using **ManifoldOptim** to results using the three algorithms in Cook and Zhang (2015) under a logistic regression. We generated 150 independent observations using $Y_i \mid X_i \sim \text{Bernoulli}(\text{logit}^{-1}(\beta^\top \text{X}_i))$, with $\beta = (\beta_1, \beta_2)^\top = (0.25, 0.25)^\top$ and $\text{X}_i \sim N(0, \Sigma_\text{X})$. We let $v_1 = \text{span}(\beta)$ be the principal eigenvector of Σ_X with eigenvalue 10 and let the second eigenvector v_2 be the orthogonal completion with eigenvalue of 0.1 such that $v_1 v_1^\top + v_2 v_2^\top = I$. This example is compelling because of the collinearity during construction of Σ_X. Consequently, this causes poor estimates when using a standard generalized linear model (GLM).

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 is performed along with Fisher scoring in an iterative fashion. Within the product manifold optimization, Γ is optimized over the Grassmann manifold while η and α are estimated over the Euclidean manifold. Numerical derivatives are calculated with **ManifoldOptim** instead of using the analytical expression from Cook and Zhang (2015) since the latter pertains only 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).

An implementation of the optimization step 2 in Algorithm 1 is shown in the following R code, where the iteration is performed ten times.

```
R> for (iter in 1:10) {
+    # perform an optimization over Grassman for gamma only
+    mani.params <- get.manifold.params(IsCheckParams = TRUE)
+    solver.params <- get.solver.params(DEBUG = 0, Tolerance = 1e-300,
+      Max_Iteration = 1000, IsCheckParams = TRUE, IsCheckGradHess = FALSE)
+    mani.defn <- get.grassmann.defn(r, u)
+    res2 <- manifold.optim(prob, mani.defn, method = "LRBFGS",
+      mani.params = mani.params, solver.params = solver.params, x0 = x0)
+    par2 <- tx(res2$xopt)
+
+    # perform a logistic regression to find eta and alpha
+    m2 <- glm(as.factor(Y) ~ (t(X) %*% par2$gammahat), family = binomial())
+
+    # update eta and alpha
+    eta <- matrix(m2$coefficients[2], nr = 1)
+    alpha <- 1 / eta
+  }
```

Alternatively, an implementation of the product manifold is shown in the following R code.

```
R> mani.params <- get.manifold.params(IsCheckParams = TRUE)
R> solver.params <- get.solver.params(DEBUG = 0, Tolerance = 1e-300,
+    Max_Iteration = 1000, IsCheckParams = TRUE, IsCheckGradHess = FALSE)
R> mani.defn <- get.product.defn(get.grassmann.defn(r, u),
+    get.euclidean.defn(u, 1), get.euclidean.defn(u, 1))
R> res <- manifold.optim(prob, mani.defn, method = "LRBFGS",
+    mani.params = mani.params, solver.params = solver.params, x0 = x0)
```

Observe the reduced code complexity when using a product manifold. To avoid local maxima, which is a problem with all optimization methods considered in this section, optimization was repeated several times from random starting points. Out of the many estimates at convergence, the one with the largest likelihood value was selected. The full code, provided as a supplement to this article, makes use of the R packages **far** (Damon and Guillas 2015), **ldr** (Adragni and Raim 2014), **MASS** (Venables and Ripley 2002), and also **mvtnorm** (Genz *et al.* 2017).

By examining the results shown in Table 5 it is immediately clear that the collinearity causes serious issues with classical methods of estimation such as GLM. The estimates in this case are not even close to the true values. However, both of the other methods perform well. Similar to Section 4.2, also note that the alternating optimization method shown in Cook's Algorithm 1 provides comparable results to the product manifold generated using **ManifoldOptim**. Since the product manifold is easier to implement, it can be a viable alternative.

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 the product space of manifolds to optimize over multiple variables where each is constrained to its own manifold. Optimization over manifold-constrained parameters is needed for 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.

Disclaimer

This article is released to inform interested parties of ongoing research and to encourage discussion of work in progress. The views expressed are those of the authors and not necessarily those of the U.S. Census Bureau.

References

Absil PA, Baker CG, Gallivan KA (2007). "Trust-Region Methods on Riemannian Manifolds." *Foundations of Computational Mathematics*, **7**(3), 303–330. doi:10.1007/s10208-005-0179-9.

Absil PA, Mahony R, Sepulchre R (2008). *Optimization Algorithms on Matrix Manifolds*. Princeton University Press, Princeton.

Adragni KP (2017). "Minimum Average Deviance Estimation for Sufficient Dimension Reduction." *Journal of Statistical Computation and Simulation*, **88**(3), 411–431. doi:10.1080/00949655.2017.1392523.

Adragni KP, Cook RD, Wu S (2012). "**GrassmannOptim**: An R Package for Grassmann Manifold Optimization." *Journal of Statistical Software*, **50**(5), 1–18. doi:10.18637/jss.v050.i05.

Adragni KP, Martin SR, Raim AM, Huang W (2020). ***ManifoldOptim**: An R Interface to the **ROPTLIB** Library for Riemannian Manifold Optimization.* R package version 1.0.0, URL https://CRAN.R-project.org/package=ManifoldOptim.

Adragni KP, Raim AM (2014). "**ldr**: An R Software Package for Likelihood-Based Sufficient Dimension Reduction." *Journal of Statistical Software*, **61**(3), 1–21. doi:10.18637/jss.v061.i03.

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*. 3rd edition. Society for Industrial and Applied Mathematics, Philadelphia.

Bailey DM, Collins MA, Gordon JDM, Zuur AF, Priede IG (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*, **276**(1664), 1–5. doi:10.1098/rspb.2009.0098.

Berry MW, Drmac Z, Jessup ER (1999). "Matrices, Vector Spaces, and Information Retrieval." *SIAM Review*, **41**(2), 335–362. doi:10.1137/s0036144598347035.

Bezanson J, Karpinski S, Shah VB, Edelman A (2012). "Julia: A Fast Dynamic Language for Technical Computing." arXiv:1209.5145 [cs.PL], URL http://arxiv.org/abs/1209.5145.

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.

Chikuse Y (2003). *Statistics on Special Manifolds*, volume 174 of *Lecture Notes in Statistics*. Springer-Verlag. doi:10.1007/978-0-387-21540-2.

Comon P, Golub GH (1990). "Tracking a Few Extreme Singular Values and Vectors in Signal Processing." *Proceedings of the IEEE*, **78**(8), 1327–1343. doi:10.1109/5.58320.

Cook RD (1998). *Regression Graphics*. John Wiley & Sons. doi:10.1002/9780470316931.

Cook RD (2007). "Fisher Lecture: Dimension Reduction in Regression." *Statistical Science*, **22**(1), 1–26. doi:10.1214/088342306000000682.

Cook RD, Forzani L (2008). "Covariance Reducing Models: An Alternative to Spectral Modelling of Covariance." *Biometrika*, **95**(4), 799–812. doi:10.1093/biomet/asn052.

Cook RD, Forzani L (2009). "Likelihood-Based Sufficient Dimension Reduction." *Journal of the American Statistical Association*, **104**(485), 197–208. doi:10.1198/jasa.2009.0106.

Cook RD, Li B, Chiaromonte F (2010). "Envelope Models for Parsimonious and Efficient Multivariate Linear Regression." *Statistica Sinica*, **20**(3), 927–1010. doi:10.5705/ss.2010.240.

Cook RD, Li L (2009). "Dimension Reduction in Regression with Exponential Family Predictors." *Journal of Computational and Graphical Statistics*, **18**(3), 774–791. doi:10.1198/jcgs.2009.08005.

Cook RD, Su Z (2013). "Scaled Envelopes: Scale-Invariant and Efficient Estimation in Multivariate Linear Regression." *Biometrika*, **100**(4), 939–954. doi:10.1093/biomet/ast026.

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.

Csiszar I, Tusnady G (1984). "Information Geometry and Alternating Minimization Procedures." *Statistics and Decisions, Supplementary Issue*, **1**, 205–237.

Damon J, Guillas S (2015). ***far**: Modelization for Functional Autoregressive Processes*. R package version 0.6-5, URL https://CRAN.R-project.org/package=far.

Duong T (2007). "**ks**: Kernel Density Estimation and Kernel Discriminant Analysis for Multivariate Data in R." *Journal of Statistical Software*, **21**(7), 1–16. doi:10.18637/jss.v021.i07.

Eddelbuettel D (2013). *Seamless R and C++ Integration with **Rcpp***. Springer-Verlag.

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.

Eddelbuettel D, François R (2016). *Exposing C++ Functions and Classes with **Rcpp** Modules*. **Rcpp** version 0.12.3 as of 2016-01-10.

Eddelbuettel D, Sanderson C (2014). "**RcppArmadillo**: Accelerating R with High-Performance C++ Linear Algebra." *Computational Statistics & Data Analysis*, **71**, 1054–1063. doi: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. doi:10.1137/s0895479895290954.

Fletcher R (1987). *Practical Methods of Optimization*. 2nd edition. John Wiley & Sons.

Genz A, Bretz F, Miwa T, Mi X, Leisch F, Scheipl F, Hothorn T (2017). ***mvtnorm**: Multivariate Normal and t Distributions*. R package version 1.0-6, URL http://CRAN.R-project.org/package=mvtnorm.

He K, He S, Huang J, Xie Y (2014). ***rOptManifold**: Optimizing Functions over Various Matrix Manifolds*. R package version 1.0, URL https://github.com/Kejun2013/rOptManifold, URL https://optmanifold.wordpress.com/.

Hilbe JM (2016). ***COUNT**: Functions, Data and Code for Count Data*. R package version 1.3.4, URL https://CRAN.R-project.org/package=COUNT.

Huang W, Absil PA, Gallivan KA (2015a). "A Riemannian Symmetric Rank-One Trust-Region Method." *Mathematical Programming*, **150**(2), 179–216. doi:10.1007/s10107-014-0765-1.

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, Absil PA, Gallivan KA, Hand P (2017). ***ROPTLIB**: Riemannian Manifold Optimization Library*. URL http://www.math.fsu.edu/~whuang2/pdf/USER_MANUAL_for_2017-02-26.pdf.

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. doi:10.1137/140955483.

Lee J (2000). *Introduction to Topological Manifolds*. 2nd edition. Springer-Verlag. doi:10.1007/b98853.

Lee J (2003). *Introduction to Smooth Manifolds.* Springer-Verlag.

Lippert R (2008). "Stiefel Grassmann Optimization (**sg_min**)." URL `https://web.archive.org/web/20080520035132/http://www-math.mit.edu/~lippert/sgmin.html`.

Louboutin M, Lange M, Herrmann F, Kukreja N, Gorman G (2017). "Performance Prediction of Finite-Difference Solvers for Different Computer Architectures." *Computers & Geosciences*, **105**, 148–157. `doi:10.1016/j.cageo.2017.04.014`.

Nocedal J, Wright S (1999). *Numerical Optimization.* Springer-Verlag. `doi:10.1007/b98874`.

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

R Core Team (2019). *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria. URL `https://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. `doi:10.1137/11082885x`.

Sanderson C, Curtin R (2016). "**Armadillo**: A Template-Based C++ Library for Linear Algebra." *Journal of Open Source Software*, **1**(2). `doi:10.21105/joss.00026`.

Sato H, Iwai T (2013). "A Riemannian Optimization Approach to the Matrix Singular Value Decomposition." *SIAM Journal on Optimization*, **23**(1), 188–212. `doi:10.1137/120872887`.

Su Z, Cook RD (2011). "Partial Envelopes for Efficient Estimation in Multivariate Linear Regression." *Biometrika*, **98**(1), 133–146. `doi:10.1093/biomet/asq063`.

Su Z, Cook RD (2012). "Inner Envelopes: Efficient Estimation in Multivariate Linear Regression." *Biometrika*, **99**(3), 687–702. `doi:10.1093/biomet/ass024`.

The MathWorks Inc (2019). *MATLAB – The Language of Technical Computing, Version R2019a.* Natick, Massachusetts. URL `http://www.mathworks.com/products/matlab/`.

Townsend J, Koep N, Weichwald S (2016). "**Pymanopt**: A Python Toolbox for Optimization on Manifolds Using Automatic Differentiation." *Journal of Machine Learning Research*, **17**(137), 1–5.

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

Van Rossum G, *et al.* (2019). *Python Programming Language.* URL `http://www.python.org`.

Venables WN, Ripley BD (2002). *Modern Applied Statistics with S.* 4th edition. Springer-Verlag, New York. URL `http://www.stats.ox.ac.uk/pub/MASS4`.

Wong YC (1967). "Differential Geometry of Grassmann Manifolds." *Proceedings of the National Academy of Sciences of the United States of America*, **57**(3), 589–594. `doi:10.1073/pnas.57.3.589`.

Xia Y, Tong H, Li WK, Zhu LX (2002). "An Adaptive Estimation of Dimension Reduction Space." *Journal of the Royal Statistical Society B*, **64**(3), 363–410. doi:10.1111/1467-9868.03411.

Affiliation:

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

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
April 2020, Volume 93, Issue 1 *Submitted:* 2017-02-12
doi:10.18637/jss.v093.i01 *Accepted:* 2018-09-09