FLORIDA STATE UNIVERSITY

COLLEGE OF ARTS AND SCIENCES

RIEMANNIAN OPTIMIZATION METHODS FOR AVERAGING SYMMETRIC POSITIVE DEFINITE MATRICES

By

XINRU YUAN

A Dissertation submitted to the
Department of Mathematics
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

2018

Xinru Yuan defended this dissertation on July 20, 2018.
The members of the supervisory committee were:

Kyle A. Gallivan

Professor Co-Directing Dissertation

Pierre-Antoine Absil

Professor Co-Directing Dissertation

Gordon Erlebacher

University Representative

Giray Okten

Committee Member

Martin Bauer

Committee Member

The Graduate School has verified and approved the above-named committee members, and certifies that the dissertation has been approved in accordance with university requirements.

ACKNOWLEDGMENTS

First and foremost, I would like to heavily thank my advisors, Kyle A. Gallivan and Pierre-Antoine Absil, for their invaluable guidance throughout my PhD journey. They are wonderful mentors and researchers. I have learned so much from them not only academic expertise but also the attitude towards research. I thank them sincerely for their substantial advice, support and encouragement.

Secondly I would like to thank Gordon Erlebacher, Giray Okten and Martin Bauer for serving my defense committees and providing me with insightful feedback. I also would like to thank Eric Klassen for attending my prospectus and valuable suggestions.

I would also like to thank Wen Huang for offering generous help and answering my numerous questions during the past five years. I would like to thank Xue Huang, my dear roommate. She is my family in Tallahassee. We have shared a lot of laughters and tears. I would also like to thank all my friends in Tallahassee for all the good times we had together.

Last but not least, I would like to give my special thanks to my family, in particular to my parents, Yin Yuan and Junhua Li, my grandparents, Huisheng Li and Yunfen Ren, for their unconditional love.

To Weiluo: for support, understanding, encouragement and love. For making me feel blessed everyday. Without his help, I would have never made it this far.

I dedicate this thesis to the memory of my grandpa, Huisheng Li.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Symmetric positive definite (SPD) matrices have become fundamental computational objects in many areas. It is often of interest to average a collection of symmetric positive definite matrices. This dissertation investigates different averaging techniques for symmetric positive definite matrices. We use recent developments in Riemannian optimization to develop efficient and robust algorithms to handle this computational task. We provide methods to produce efficient numerical representations of geometric objects that are required for Riemannian optimization methods on the manifold of symmetric positive definite matrices. In addition, we offer theoretical and empirical suggestions on how to choose between various methods and parameters. In the end, we evaluate the performance of different averaging techniques in applications.

CHAPTER 1

INTRODUCTION

1.1 Motivation and problem

Symmetric positive definite (SPD) matrices have become fundamental computational objects in many areas. For example, they appear as diffusion tensors in medical imaging [23, 34, 81], as data covariance matrices in radar signal processing [13, 62], and as elasticity tensors in elasticity [71]. In these and similar applications, it is often of interest to average or find a representative for a collection of SPD matrices. Averaging is required, e.g., to aggregate several noisy measurements of the same object. It also appears as a subtask in interpolation methods [1] and segmentation [14,78]. In clustering methods, finding a cluster center as a representative of each cluster is crucial. Hence it is desirable to find a center that is intrinsically representative and can be computed efficiently.

The Karcher mean proposed in [56] has been recognized as one of the most suitable means for SPD matrices as it holds a list of desired properties. It is defined as the minimizer to an optimization problem on the manifold of SPD matrices. Various methods have been used to compute the Karcher mean, and most of them resort to the framework of Riemannian optimization, see [16,52,53,82,83]. We revisit this problem because recently there are substantial developments on the theory and efficient implementations in the field of Riemannian optimization. At this point, we are able to provide theoretical explanations to what was observed in the literature. Moreover, we can go beyond the state-of-the-art by using recent advances from [46, 48, 49, 50, 51, 96] to develop a more efficient and robust approach for Karcher mean computation.

Even though the Karcher mean is attractive from the theoretical point of view, its computational cost increases dramatically with the size of the SPD matrices. This motivates us to investigate other definitions of matrix means based on information-theoretic divergences. A divergence is similar to a distance which also provides a measure of dissimilarity between two elements. Since several divergence functions on the set of SPD matrices have been discussed in the literature, an extensive overview of the divergences is presented, along with a summary of related means and their properties. Then we employ our Riemannian optimization techniques on the manifold of

SPD matrices to compute the divergence-based means more efficiently than the state-of-the-art approach.

The mean provides an intuitive central representative of a collection of elements, which is, however, sensitive to outliers. A median is more robust against outliers than a mean. Therefore, another contribution of our work is to address the problem of defining and computing the median of a collection of SPD matrices.

Important applications of averaging are found in the supervised classification and unsupervised clustering tasks. We evaluate and compare the performance of different averaging techniques in real world applications as well as synthetic datasets. We also contribute a C++ toolbox for different averaging techniques using a large number of optimization algorithms.

1.2 Overview and dissertation statement

This dissertation investigates different averaging techniques and similarity measures for SPD matrices. We propose to use recent developments in Riemannian optimization to develop efficient and robust algorithms to compute different central representatives of a collection of SPD matrices; to understand the state-of-the-art methods and provide theoretical explanations for the numerical observations in the literature; to provide a C++ toolbox to compute matrix means, medians, and minimax centers using various Riemannian optimization methods; to provide user guidelines on how to choose between various methods and parameters; to evaluate the performance of different averaging techniques in applications.

This dissertation asserts that the proposal above can be achieved by the following:

1. An analysis on the conditioning of the Riemannian and Euclidean Hessians of the cost function on the manifold of SPD matrices (Chapter 2)

2. The development of efficient numerical representations of geometric objects that are required for Riemannian optimization methods on the manifold of SPD matrices to improve the performance of state-of-the-art algorithms (Chapter 2)

3. The use of a limited-memory Riemannian BFGS method to reduce storage requirements and computation time (Chapter 2)

4. A computational complexity analysis on problem-related, manifold-related, and algorithm-related operations (Chapter 2, 3)

5. Investigating other definitions of means for SPD matrices based on information-theoretic divergences (Chapter 3)

6. A proof of the geodesic convexity of the log-determinant α-divergence (Chapter 3)

7. For the log-determinant α-divergence-based mean computation, cast the state-of-the-art fixed point algorithm into a Riemannian steepest descent for a choice of the cost function, retraction, and stepsize strategy (Chapter 3)

8. A numerical illustration of the relationship between the Barzilai-Borwein stepsizes and the eigenvalues of the Riemannian Hessian of the objective function (Chapter 3)

9. Systematic numerical experiments to compare and evaluate the performance of various optimization algorithms (Chapter 2, 3, 4, 5)

10. Tackling the problem of finding the Riemannian median and minimax center of a collection of SPD matrices (Chapter 4, 5)

11. The use of the modified Riemannian quasi-Newton algorithms and the nonsmooth quasi-Newton algorithms to handle nonsmooth functions in the median computation and minimax center computation problems (Chapter 4, 5)

12. The use of applications to evaluate the performance of different averaging techniques in supervised classification and unsupervised clustering (Chapter 6)

1.3 Dissertation outline

This dissertation is organized as follows.

Chapter 1. The remainder of Chapter 1 reviews some important definitions and concepts for Riemannian manifolds and optimization algorithms.

Chapter 2. This chapter addresses the problem of computing the Karcher mean of a collection of SPD matrices. We start with an analysis on the conditioning of the problem, and provide theoretical explanations for numerical observations in the literature. Then we propose to use recent developments in Riemannian optimization to develop efficient and robust algorithms for Karcher mean computation.

CHAPTER 3. This chapter investigates other definitions of means for SPD matrices based on information-theoretic divergences. We study the properties of these means, and apply our Riemannian optimization techniques developed in Chapter 2 to compute them and outperforms the state-of-the-art fixed-point method. Moreover, we cast the fixed-point algorithm into Riemannian optimization framework.

CHAPTER 4. This chapter tackles the problem of finding the median of a collection of SPD matrices based on the Riemannian geodesic distance and the log-determinant α-divergence. We exploit the modified Riemannian quasi-Newton algorithms and the nonsmooth quasi-Newton algorithms to handle this computational task.

CHAPTER 5. This chapter handles the problem of computing the minimax center of a collection of SPD matrices based on the Riemannian geodesic distance and the log-determinant α-divergence.

CHAPTER 6. This chapter is devoted to applications that require averaging SPD matrices, and we focus on the supervised classification and unsupervised clustering tasks. In the supervised scenario, we revisit the Electroencephalography (EEG) classification problem using the Minimum Distance to Mean (MDM) classifier. In the unsupervised case, we consider the problem of material categorization using K-means clustering.

CHAPTER 7. This chapter gives a summary of completed work.

1.4 Basic principles for manifolds

This section reviews some important concepts and definitions that are extensively used in the dissertation, see also [2].

1.4.1 Optimization on a manifold

Optimization on Riemannian manifolds, also called Riemannian optimization, addresses the problem of finding an optimum of a real-valued function f defined on a Riemannian manifold, i.e.,

$$\min_{x \in \mathcal{M}} f(x), \tag{1.4.1}$$

where $\mathcal{M}$ is a Riemannian manifold. Roughly speaking, a d-dimensional manifold is a set that is locally smoothly identified with open subsets of Euclidean space $\mathbb{R}^d$.

1.4.2 Tangent vector and tangent space

In order to apply line search algorithms, we must consider the direction of motion on a manifold. Let $\gamma(t) : \mathbb{R} \to \mathcal{M} : t \mapsto \gamma(t)$ be a smooth mapping on $\mathcal{M}$ satisfying $\gamma(0) = x$. That is, γ is a curve through x at $t = 0$. Given a smooth real-valued function f on $\mathcal{M}$, the function $f \circ \gamma : t \mapsto f(\gamma(t))$ is a smooth function from $\mathbb{R}$ to $\mathbb{R}$ with a well-defined classical derivative. Let $\mathcal{F}_x(\mathcal{M})$ denote the set of smooth real-valued functions on a neighborhood of x. We can define the mapping $\dot{\gamma}(0)$ from $\mathcal{F}_x(\mathcal{M})$ to $\mathbb{R}$ by

$$\begin{aligned} \dot{\gamma}(0) &= (f \circ \gamma)'(0) \\ &= \lim_{h \to 0} \frac{f(\gamma(h)) - f(\gamma(0)))}{h}. \end{aligned} \tag{1.4.2}$$

This mapping is a tangent vector to the curve γ at $t = 0$ and it defines the direction at x along γ. The formal definition of tangent vector is as follows.

Definition 1.4.1 (tangent vector)**.** *A tangent vector ξ_x to a manifold $\mathcal{M}$ at a point x is a mapping from $\mathcal{F}_x(\mathcal{M})$ to $\mathbb{R}$ such that there exists a curve γ on $\mathcal{M}$ with $\gamma(0) = x$, satisfying*

$$\xi_x f = \dot{\gamma}(0) f := \left. \frac{d(f(\gamma(t)))}{dt} \right|_{t=0} \tag{1.4.3}$$

for all $f \in \mathcal{F}_x(\mathcal{M})$. The curve γ is said to realize the tangent vector ξ_x. The point x is called the root of the tangent vector ξ_x.

The tangent space to $\mathcal{M}$ at x, denoted by $\mathrm{T}_x \mathcal{M}$, is the set of all tangent vectors at x. The tangent space is a vector space, and has the same dimension as the manifold. So we perform line search on the tangent space, and use retraction (see Section 1.4.5) to get back to the manifold. The union of all tangent spaces is called the tangent bundle of the manifold, denoted by $T\mathcal{M}$.

Another important concept is the vector field. A vector field ξ on a manifold $\mathcal{M}$ is a smooth function from $\mathcal{M}$ to the tangent bundle $\xi : \mathcal{M} \to T\mathcal{M} : x \mapsto \xi_x \in \mathrm{T}_x \mathcal{M}$. It assigns to each point a tangent vector.

1.4.3 Riemannian metric

The tangent space at a point on the manifold provides us with a vector space that approximates the manifold locally. Endowing the tangent space with an inner product allows us to compute angles and lengths of tangent vectors.

Definition 1.4.2 (inner product)**.** *Let $\mathcal{M}$ be a smooth manifold and $x \in \mathcal{M}$. An inner product $\langle \cdot, \cdot \rangle_x$ on $\mathrm{T}_x \mathcal{M}$ is a bilinear, symmetric positive-definite form, i.e., $\forall \xi, \zeta, \eta \in \mathrm{T}_x \mathcal{M}$, $a, b \in \mathbb{R}$, $\langle a\xi + b\zeta, \eta \rangle_x = a\langle \xi, \eta \rangle_x + b\langle \zeta, \eta \rangle_x$, $\langle \xi, \zeta \rangle_x = \langle \zeta, \xi \rangle_x$, and $\langle \xi, \xi \rangle_x \geq 0$ with $\langle \xi, \xi \rangle_x = 0 \Leftrightarrow \xi = 0$.*

A Riemannian metric g defined on the tangent spaces of x is a smoothly varying inner product $g_x : \mathrm{T}_x \mathcal{M} \times \mathrm{T}_x \mathcal{M} \to \mathbb{R}$. We will use interchangeably the notation

$$g_x(\xi, \eta) = \langle \xi, \eta \rangle_x \tag{1.4.4}$$

to denote the Riemannian metric, where $\xi, \eta \in \mathrm{T}_x \mathcal{M}$ and the subscript x is dropped when clear from the context. A notation, flat $\flat$, is also used in later sections. We define $\xi^\flat$ as a function from $\mathrm{T}_x \mathcal{M}$ to $\mathbb{R}$ such that $\xi^\flat \eta = g_x(\xi, \eta)$ for all $\eta \in \mathrm{T}_x \mathcal{M}$. A Riemannian manifold is the combination $(\mathcal{M}, g)$.

The length of a curve $\gamma : [0, 1] \to \mathcal{M}$ on a Riemannian manifold $(\mathcal{M}, g)$ is defined by

$$d(x, y) = \int_0^1 \sqrt{g_{\gamma(t)}(\dot{\gamma}(t), \dot{\gamma}(t))} dt = \int_0^1 \|\dot{\gamma}(t)\|_{g_{\gamma(t)}} dt. \tag{1.4.5}$$

The Riemannian distance on the manifold is

$$d(x, y) = \inf_\gamma L(\gamma), \tag{1.4.6}$$

where γ is a curve on $\mathcal{M}$ with $\gamma(0) = x$ and $\gamma(1) = y$.

1.4.4 Affine connections, geodesics, exponential mapping and parallel translation

In Euclidean space $\mathbb{R}^n$, straight lines are curves γ with zero acceleration, i.e.,

$$\frac{d^2}{dt^2} \gamma(t) = 0, \text{ for all } t.$$

Geodesics on manifolds generalize the concept of straight lines in $\mathbb{R}^n$. In order to define acceleration on manifolds, we need the notion of affine connection, which provides the idea of differentiating tangent vectors.

Definition 1.4.3 (affine connection)**.** *Let $\mathcal{F}_x(\mathcal{M})$ denote the set of all smooth functions on a neighborhood of x, and $\mathcal{X}(\mathcal{M})$ denote the set of smooth vector fields on $\mathcal{M}$. An affine connection ∇ on a manifold $\mathcal{M}$ is a mapping*

$$\nabla : \mathcal{X}(\mathcal{M}) \times \mathcal{X}(\mathcal{M}) \to \mathcal{X}(\mathcal{M}) : (\xi, \eta) \mapsto \nabla_\xi \eta \tag{1.4.7}$$

that satisfies the following properties: for all $f, g \in \mathcal{F}_x(\mathcal{M})$, $a, b \in \mathbb{R}$, and $\eta, \xi, \zeta \in \mathcal{X}(\mathcal{M})$:

1. *$\mathcal{F}(\mathcal{M})$-linearity in the first argument:* $\nabla_{f\eta+g\zeta}\xi = f\nabla_\eta\xi + g\nabla_\zeta\xi$;

2. *$\mathbb{R}$-linearity in the second argument:* $\nabla_\eta(a\xi + b\zeta) = a\nabla_\eta\xi + b\nabla_\eta\zeta$;

3. *Product rule (Leibniz's law):* $\nabla_\eta(f\xi) = (\eta f)\xi + f\nabla_\eta\xi$.

The resulting vector field $\nabla_\xi\eta$ is called the covariant derivative of ξ with respect to η for the affine connection ∇.

Remark 1.4.1. *1. ηf denotes the application of the vector field η to the function f defined by $(\eta f)(x) := \eta_x f$; 2. The multiplication of a vector field ξ by a function f is defined by $(f\xi)_x := f(x)\xi_x$; 3. The addition of two vector fields is defined by $(\xi + \zeta)_x := \xi_x + \zeta_x$ for all $x \in \mathcal{M}$.*

Any manifold $\mathcal{M}$ admits an infinite number of affine connections. However, there are certain affine connections that may be preferred due to particular properties. On a Riemannian manifold $(\mathcal{M}, g)$, a preferred affine connection, called the Riemannian connection or Levi-Civita connection, satisfies the following two additional conditions:

1. symmetry: $(\nabla_\eta\xi - \nabla_\xi\eta)f = \eta(\xi f) - \xi(\eta f)$;

2. compatibility with Riemannian metric: $\zeta g(\eta, \xi) = g(\nabla_\zeta\eta, \xi) + g(\eta, \nabla_\zeta\xi)$.

A curve γ on a Riemannian manifold $(\mathcal{M}, g)$ endowed with an affine connection ∇ is a geodesic if it has zero acceleration:

$$\nabla_{\dot\gamma(t)}\dot\gamma(t) := \frac{d^2}{dt^2}\gamma(t) := \frac{d}{dt}\dot\gamma(t) = 0 \tag{1.4.8}$$

for all t. With the Riemannian connection, one of the geodesics linking two points on the manifold is also a minimal length curve. In this dissertation, we only consider the Riemannian connection.

Given a point $x \in \mathcal{M}$ and a tangent vector $\eta \in \mathrm{T}_x\,\mathcal{M}$, there exists a unique geodesic $\gamma(t; x, \eta)$ satisfying $\gamma(0) = x$ and $\dot\gamma(0) = \eta$. In addition, the geodesic has the homogeneity property, $\gamma(t; x, a\eta) = \gamma(at; x, \eta)$. The mapping

$$\mathrm{Exp}_x : \mathrm{T}_x\,\mathcal{M} \to \mathcal{M} : \eta \mapsto \mathrm{Exp}_x\,\eta = \gamma(1; x, \eta), \tag{1.4.9}$$

is called the exponential mapping at x. A manifold $(\mathcal{M}, g)$ is called geodesically complete if and only if Exp_x is defined for all $x \in \mathcal{M}$ and all $\eta \in \mathrm{T}_x\,\mathcal{M}$. That is, every geodesic of a geodesically

complete manifold can be extended indefinitely. When performing line search algorithms, exponential mapping allows us to move in the direction of a tangent vector in the tangent space, and then map the tangent vector to a point on the manifold.

A related concept is the log-mapping. It is defined as the inverse of the exponential mapping

$$\mathrm{Exp}_x^{-1} : \mathcal{M} \to \mathrm{T}_x\mathcal{M} : y \mapsto \mathrm{Exp}_x^{-1}(y) = \eta, \tag{1.4.10}$$

where the geodesic curve $t \mapsto \gamma(t)$ with $\gamma(0) = x$ and $\dot{\gamma}(0) = \eta$ satisfies $\gamma(1) = y$. We may also use notation Log_x to denote the log-mapping.

In many situations, we may need to compare or combine tangent vectors at different points on the manifold, which are in different tangent spaces. So we need to "transport" them to a common tangent space. The affine connection can be used to define the notion of moving a tangent vector from one tangent space to another, called parallel translation.

A vector field ξ on a curve γ that satisfies $\frac{d}{dt}\xi = \nabla_{\dot{\gamma}}\xi = 0$ is called parallel. Give $a \in \mathbb{R}$ in the domain of γ and $\xi_{\gamma(a)} \in \mathrm{T}_{\gamma(a)}\mathcal{M}$, there exists a unique parallel vector field ξ on γ such that $\xi(a) = \xi_{\gamma(a)}$. The operator $P_\gamma^{b\leftarrow a}$ sending $\xi(a)$ to $\xi(b)$ is called parallel translation along γ. In other words, we have

$$\frac{d}{dt}(P_\gamma^{t\leftarrow a}\xi(a)) = 0. \tag{1.4.11}$$

If ∇ is the Riemannian connection, the parallel translation is an isometry, i.e.,

$$\langle P_\gamma^{t\leftarrow a}\xi(a), P_\gamma^{t\leftarrow a}\zeta(a)\rangle = \langle \xi(a), \zeta(a)\rangle.$$

1.4.5 Retraction and vector transport

A retraction is a smooth mapping that maps a tangent vector to a point on the manifold. That is, we perform line search on the tangent space, and use retraction to get back to the manifold to obtain the next iterate. A retraction allows us to move in the direction of a tangent vector while staying on the manifold. The exponential mapping is a special retraction. When the exponential mapping is used to map a tangent vector back to the manifold, we actually move along the geodesic defined by the tangent vector. The formal definition of retraction follows.

Definition 1.4.4 (retraction)**.** *A retraction on a manifold $\mathcal{M}$ is a smooth mapping R from the tangent bundle $\mathrm{T}\mathcal{M}$ onto $\mathcal{M}$ with the following properties.*

1. $R(0_x) = x$ *for all* $x \in \mathcal{M}$*, where* 0_x *denotes the zero element of* $\mathrm{T}_x\,\mathcal{M}$*.*
2. $\frac{d}{dt}R(t\xi_x)|_{t=0} = \xi_x$ *for all* $\xi_x \in \mathrm{T}_x\,\mathcal{M}$*.*

The restriction of R *to* $\mathrm{T}_x\,\mathcal{M}$ *is denoted by* R_x*.*

A vector transport is a mapping that transports a tangent vector from one tangent space to another tangent space. It is a more general concept related to parallel translation along geodesics. Much like the exponential mapping, the parallel translation is often computationally demanding. Vector transport provides an alternative to parallel translation, and may reduce the computational cost.

Definition 1.4.5 (vector transport)**.** *A vector transport on a manifold* $\mathcal{M}$ *is a smooth mapping*

$$\mathcal{T} : \mathrm{T}\,\mathcal{M} \oplus \mathrm{T}\,\mathcal{M} \to \mathrm{T}\,\mathcal{M},\ (\eta_x, \xi_x) \mapsto \mathcal{T}_{\eta_x}\xi_x$$

satisfying the following properties for all $x \in \mathcal{M}$*;*

1. *(Associated retraction) There exists a retraction* R*, called the retraction associated with* $\mathcal{T}$*, such that the following diagram commutes*

$$\begin{array}{ccc} (\eta_x,\, \xi_x) & \xrightarrow{\quad\mathcal{T}\quad} & \mathcal{T}_{\eta_x}(\xi_x) \\ \big\downarrow & & \big\downarrow \pi \\ \eta_x & \xrightarrow[\quad R\quad]{} & \pi(\mathcal{T}_{\eta_x}(\xi_x)) \end{array} \tag{1.4.12}$$

   where $\pi(\mathcal{T}_{\eta_x}(\xi_x))$ *denotes the foot of the tangent vector* $\mathcal{T}_{\eta_x}(\xi_x)$*.*
2. *(Consistency)* $\mathcal{T}_{0_x}\xi_x = \xi_x$ *for all* $\xi_x \in \mathrm{T}_x\,\mathcal{M}$*;*
3. *(Linearity)* $\mathcal{T}_{\eta_x}(a\xi_x + b\zeta_x) = a\mathcal{T}_{\eta_x}(\xi_x) + b\mathcal{T}_{\eta_x}(\zeta_x)$*.*

A vector transport is called isometric if it satisfies

$$\langle \mathcal{T}_{\eta_x}\xi_x, \mathcal{T}_{\eta_x}\zeta_x \rangle_{R_{\eta_x}} = \langle \xi_x, \zeta_x \rangle_x. \tag{1.4.13}$$

Given a retraction R on a manifold $\mathcal{M}$, vector transport by differentiated retraction is an important approach to produce vector transport, which is given by

$$\begin{aligned} \mathcal{T}_{\eta_x}\xi_x &= \mathrm{D}\,R_x(\eta_x)[\xi_x] \\ &= \frac{d}{dt}\,R_x(\eta_x + t\xi_x)|_{t=0}\,. \end{aligned} \tag{1.4.14}$$

The choice of retraction and vector transport is a key step in the design of efficient Riemannian optimization algorithms, which we will see in later sections.

1.4.6 Riemannian gradient and Hessian

The gradient of a function gives the direction in which the function increases most rapidly, and is proved to be useful for optimization methods in an Euclidean space. The gradient of a function on a Riemannian manifold is a tangent vector, which is defined as follows.

Definition 1.4.6 (Riemannian gradient)**.** *Let f be a function defined on a Riemannian manifold $(\mathcal{M}, g)$. The Riemannian gradient of f at $x \in \mathcal{M}$, denoted by* $\operatorname{grad} f$*, is the unique tangent vector that satisfies*

$$\langle \operatorname{grad} f(x), \xi \rangle_x = \mathrm{D}\, f(x)[\xi], \quad \forall \xi \in \mathrm{T}_x \mathcal{M}\,. \tag{1.4.15}$$

The element $Df(x)[\xi]$ is called the directional derivative of f at x along ξ.

Second-order optimization algorithms, such as Newton's method, may require the Hessian. For a real-valued function f defined on the Euclidean space $\mathbb{R}^n$, its Hessian matrix is a square matrix whose elements are second-order partial derivatives of f, i.e., $\operatorname{Hess} f(x) = (\partial^2_{ij} f(x))$. Consider the directional derivative of $\operatorname{grad} f(x)$ along direction v

$$\lim_{h\to} \frac{\operatorname{grad} f(x + tv) - \operatorname{grad} f(x)}{t} = \operatorname{Hess} f(x)[v].$$

That is, we can view the Hessian as an operator acting on v. This idea is used to formalize the Hessian on a manifold.

Definition 1.4.7 (Riemannian Hessian)**.** *Let f be a real-valued function defined on a Riemannian manifold $(\mathcal{M}, g)$, the Riemannian Hessian of f at $x \in \mathcal{M}$ is a linear mapping, denoted by* $\operatorname{Hess} f(x)$*, from $\mathrm{T}_x \mathcal{M}$ to $\mathrm{T}_x \mathcal{M}$ defined by*

$$\operatorname{Hess} f(x)[\xi_x] = \nabla_{\xi_x} \operatorname{grad} f$$

for all ξ_x in $\mathrm{T}_x \mathcal{M}$, where ∇ is the Riemannian connection on $\mathcal{M}$.

From the symmetry of the Riemannian connection, we know the Hessian is a self-adjoint operator in terms of Riemannian metric, i.e.,

$$\langle \operatorname{Hess} f(x)[\eta], \xi \rangle_x = \langle \eta, \operatorname{Hess} f(x)[\xi] \rangle,$$

for all η, $\xi \in \mathrm{T}_x \mathcal{M}$.

1.4.7 Geodesic convexity

Convexity plays an important role in Euclidean optimization. Geodesic convexity, also termed as g-convexity, generalizes the notion of convexity from linear space to nonlinear Riemannian manifold. Here we review some important definitions that can be found in [80, 98].

Definition 1.4.8. *(geodesically convex set) Let $(\mathcal{M}, g)$ be a Riemannian manifold. A subset $S \subset \mathcal{M}$ is said to be geodesically convex if any two points of S are joined by a geodesic belonging to S, i.e., for all $x, y \in S$, there exists a geodesic curve $\gamma : [0, 1] \to S$ such that $\gamma(0) = x$ and $\gamma(1) = y$.*

Definition 1.4.9. *(geodesically convex function) Let $(\mathcal{M}, g)$ be a Riemannian manifold. A function $f : \mathcal{M} \to \mathbb{R}$ is said to be geodesically convex if for any $x, y \in \mathcal{M}$, a geodesic γ such that $\gamma_1(0) = x_1$ and $\gamma(1) = y$, and $t \in [0, 1]$, it holds that*

$$f(\gamma(t)) \leq (1 - t)f(x) + tf(y) \tag{1.4.16}$$

An equivalent definition is that for any $x, y \in \mathcal{M}$,

$$f(y) \geq f(x) + \langle \operatorname{grad} f(x), \operatorname{Exp}_x^{-1}(y) \rangle_x. \tag{1.4.17}$$

Definition 1.4.10. *(jointly geodesically convex function) Let $(\mathcal{M}, g)$ be a Riemannian manifold. A function $f : \mathcal{M} \times \mathcal{M} \to \mathbb{R}$ is said to be jointly geodesically convex if for any $x_1, x_2, y_1, y_2 \in \mathcal{M}$, geodesics γ_x and γ_y such that $\gamma_x(0) = x_1$, $\gamma_x(1) = x_2$, $\gamma_y(0) = y_1$ and $\gamma_y(1) = y_2$, and $t \in [0, 1]$, it holds that*

$$f(\gamma_x(t), \gamma_y(t)) \leq (1 - t)f(x_1, y_1) + tf(x_2, y_2). \tag{1.4.18}$$

Definition 1.4.11. *(Lipschitzness) Let $(\mathcal{M}, g)$ be a Riemannian manifold. A function $f : \mathcal{M} \to \mathbb{R}$ is said to be geodesically L_f-Lipschitz if for any $x, y \in \mathcal{M}$,*

$$|f(x) - f(y)| \leq L_f \operatorname{dist}(x, y), \tag{1.4.19}$$

where dist is the Riemannian distance on $\mathcal{M}$ and L_f is a positive real number.

Definition 1.4.12. *(smoothness) Let $(\mathcal{M}, g)$ be a Riemannian manifold. A function $f : \mathcal{M} \to \mathbb{R}$ is said to be geodesically L_g-smooth if its gradient is L_g-Lipschitz, i.e., for any $x, y \in \mathcal{M}$,*

$$\| \operatorname{grad} f(x) - P_\gamma^{x \leftarrow y} \operatorname{grad} f(y) \| \leq L_g \operatorname{dist}(x, y), \tag{1.4.20}$$

where $P_\gamma^{x \leftarrow y}$ is the parallel translation from y to x and Lg is a positive real number.

1.5 Geometry of $\mathcal{S}^n_{++}$

In this dissertation, we focus on solving problems on the manifold of $n \times n$ symmetric positive definite matrices. In this section, we review the geometry of this manifold.

Let $\mathcal{S}^n$ be the set of symmetric $n \times n$ matrices

$$\mathcal{S}^n = \{A \in \mathbb{R}^{n \times n} | A = A^T\}, \tag{1.5.1}$$

and $\mathcal{S}^{\mathrm{n}}_{++}$ be the set of symmetric positive definite $n \times n$ matrices

$$\mathcal{S}^{\mathrm{n}}_{++} = \{A \in \mathbb{R}^{n \times n} | A = A^T, A > 0\}. \tag{1.5.2}$$

Here $A > 0$ denotes that the quadratic form $x^T A x > 0$ for all non zero vectors $x \in \mathbb{R}^n$.

There are two important definitions related to elements in $\mathcal{S}^{\mathrm{n}}_{++}$:

- Matrix exponential: for any $n \times n$ matrix A, its exponential, denoted by e^A or $\exp(A)$, is given by

$$e^A = \sum_{k=0}^{\infty} \frac{1}{k!} A^k. \tag{1.5.3}$$

  If $A \in \mathcal{S}^{\mathrm{n}}_{++}$, then it can factored as

$$A = Q \Lambda Q^T, \tag{1.5.4}$$

  where Q is orthogonal and Λ is diagonal. In this case, we have

$$e^A = Q e^{\Lambda} Q^T. \tag{1.5.5}$$

  Since Λ is diagonal, its exponential is simply an diagonal matrix with diagonal elements equal to the exponential of the diagonal elements of Λ.

- Matrix logarithm: the matrix logarithm is defined as the inverse of matrix exponential. Even though it is not always well defined for arbitrary matrices, it always exists and is unique for symmetric positive definite matrices.

  For a symmetric positive definite matrix $A = Q \Lambda Q^T$, its logarithm is given by

$$A = Q \log(\Lambda) Q^T, \tag{1.5.6}$$

  where $\log(\Lambda)$ is a diagonal matrix with diagonal elements equal to the logarithm of the diagonal elements of Λ. Note that since A is positive definite, the diagonal elements of Λ are positive and their logarithms are always well defined.

It is easy to verify that the matrix exponential map $\exp : \mathcal{S}^n \to \mathcal{S}^{\mathrm{n}}_{++} : A \mapsto e^A$ is one-to-one and onto. That is, for any symmetric matrix A, with eigenvalue decomposition $A = Q\Lambda Q^T$, its exponential $e^A = Qe^{\Lambda}Q^T$ is symmetric positive definite since $e^{\Lambda} > 0$. On the other hand, for any symmetric positive definite matrix B, there exists a unique symmetric matrix X such that $e^X = B$, i.e., $X = \log(B)$.

Since $\mathcal{S}^n_{++}$ is an open subset of the vector space $\mathcal{S}^n$, its tangent space at point X—denoted by $\mathrm{T}_X\,\mathcal{S}^n_{++}$—can be identified with $\mathcal{S}^n$. A typical Riemannian metric on $\mathrm{T}_X\,\mathcal{S}^{\mathrm{n}}_{++}$ is the Euclidean metric inherited from $\mathcal{S}^n$, given by

$$g_X(\xi_X, \eta_X) = \mathrm{tr}(\xi_X^T \eta_X), \tag{1.5.7}$$

where $\xi_X,\ \eta_X \in \mathrm{T}_X\,\mathcal{S}^{\mathrm{n}}_{++}$. Under the Euclidean metric, the geodesic emanating from $X \in \mathcal{S}^{\mathrm{n}}_{++}$ in the direction of $\eta_X \in \mathrm{T}_X\,\mathcal{S}^{\mathrm{n}}_{++}$ is given by

$$\gamma(t) = X + t\eta_X,\ t \in [0, \infty). \tag{1.5.8}$$

The geodesic between two matrices $X,\ Y \in \mathcal{S}^{\mathrm{n}}_{++}$ is

$$\gamma(t) = X + t(Y - X),\ t \in [0, 1] \tag{1.5.9}$$

and the geodesic distance between X and Y is

$$\delta(X, Y) = \|X - Y\|_F. \tag{1.5.10}$$

However, $\mathcal{S}^{\mathrm{n}}_{++}$ endowed with the Euclidean metric is not geodesically complete.

A more suitable Riemannian metric for $\mathcal{S}^{\mathrm{n}}_{++}$ is proposed in [78], called the affine-invariant metric, given by

$$g_X(\xi_X, \eta_X) = \mathrm{trace}(\xi_X X^{-1} \eta_X X^{-1}). \tag{1.5.11}$$

Under the affine-invariant metric (2.1.1), the geodesic $\gamma(t)$ such that $\gamma(0) = X$ and $\dot{\gamma}(0) = \eta_X$ is given by

$$\gamma(t) = X^{1/2} \exp(tX^{1-/2} \eta_X X^{-1/2}) X^{1/2}. \tag{1.5.12}$$

The geodesic between two matrices $X,\ Y \in \mathcal{S}^{\mathrm{n}}_{++}$ is

$$\gamma(t) = X^{1/2} (X^{-1/2} Y X^{-1/2})^t X^{1/2}, \tag{1.5.13}$$

and the geodesic distance between X and Y is

$$\delta(X, Y) = \| \log(X^{-1/2} Y X^{-1/2}) \|. \tag{1.5.14}$$

Notice that geodesic (1.5.12) can be extended indefinitely, and $\mathcal{S}^{\mathrm{n}}_{++}$ is geodesically complete. Also notice from (1.5.14) that the distance between any SPD matrices and symmetric matrices with null or negative eigenvalues is infinite.

Under the affine-invariant metric (2.1.1), the Riemannian connection is given by [53]

$$\nabla_{\zeta_X} \xi = \mathrm{D}(\xi)(X)[\zeta_X] - \frac{1}{2}(\zeta_X X^{-1} \xi + \xi X^{-1} \zeta_X). \tag{1.5.15}$$

CHAPTER 2

KARCHER MEAN COMPUTATION ON $\mathcal{S}^n_{++}$

This chapter addresses the problem of computing the Karcher mean of a collection of symmetric positive definite (SPD) matrices. A condensed version of this chapter can be found in [96].

2.1 Introduction

A natural way to average over a collection of SPD matrices, $\{A_1, \ldots, A_K\}$, is to take their arithmetic mean, i.e., $G(A_1, \ldots, A_K) = (A_1 + \cdots + A_K)/K$. However, this is not appropriate in applications where invariance under inversion is required, i.e., $G(A_1, \ldots, A_K)^{-1} = G(A_1^{-1}, \ldots, A_K^{-1})$. In addition, the arithmetic mean may cause a "swelling effect" that should be avoided in diffusion tensor imaging. Swelling is defined as an increase in the matrix determinant after averaging, see [34] for example. An alternative is to generalize the definition of geometric mean from scalars to matrices, which yields $G(A_1, \ldots, A_K) = (A_1 \ldots A_K)^{1/K}$. However, this generalized geometric mean is not invariant under permutation since matrices are not commutative in general. Ando et al. [6] introduced a list of fundamental properties, referred to as the ALM list, that a matrix "geometric" mean should possess:

- P1 Consistency with scalars. If $A_1, \ldots, A_K$ commute then $G(A_1, \ldots, A_K) = (A_1 \cdots A_K)^{1/K}$.
- P2 Joint homogeneity. $G(\alpha_1 A_1, \ldots, \alpha_K A_K) = (\alpha_1 \cdots \alpha_K)^{1/K} G(A_1, \ldots, A_K)$.
- P3 Permutation invariance. For any permutation $\pi(A_1, \ldots, A_K)$ of $(A_1, \ldots, A_K)$, $G(A_1, \ldots, A_K) = G(\pi(A_1, \ldots, A_K))$.
- P4 Monotonicity. If $A_i \geq B_i$ for all i, then $G(A_1, \ldots, A_K) \geq G(B_1, \ldots, B_K)$ in the positive semidefinite ordering.
- P5 Continuity from above. If $\{A_1^{(n)}\}, \ldots, \{A_k^{(n)}\}$ are monotonic decreasing sequences (in the positive semidefinite ordering) converging to A_1, ..., A_K, respectively, then $G(A_1^{(n)}, \ldots, A_K^{(n)})$ converges to $G(A_1, \ldots, A_K)$.
- P6 Congruence invariance. $G(S^T A_1 S, \ldots, S^T A_K S) = S^T G(A_1, \ldots, A_K) S$ for any invertible S.
- P7 Joint concavity. $G(\lambda A_1 + (1-\lambda)B_1, \ldots, \lambda A_K + (1-\lambda)A_K) \geq \lambda G(A_1, \ldots, A_K) + (1-\lambda) G(B_1, \ldots, B_K)$.

P8 Invariance under inversion. $G(A_1, \ldots, A_K)^{-1} = G(A_1^{-1}, \ldots, A_K^{-1})$.

P9 Determinant identity. $\det G(A_1, \ldots, A_K) = (\det A_1 \cdots \det A_K)^{1/K}$.

These properties are known to be important in numerous applications, e.g. [15,63,71]. However, they do not uniquely define a mean for $K \geq 3$. There can be many different definitions of means that satisfy all the properties. The Karcher mean proposed in [56] has been recognized as one of the most suitable means for SPD matrices in the sense that it satisfies all properties in the ALM list [15, 63].

Karcher mean. Let $\mathcal{S}_{++}^n$ be the manifold of $n \times n$ SPD matrices. Since $\mathcal{S}_{++}^n$ is an open submanifold of the vector space of $n \times n$ symmetric matrices, its tangent space at point X, denoted by $\mathrm{T}_X \mathcal{S}_{++}^n$, can be identified as the set of $n \times n$ symmetric matrices. The manifold $\mathcal{S}_{++}^n$ becomes a Riemannian manifold when endowed with the affine-invariant metric, see [78], given by

$$g_X(\xi_X, \eta_X) = \mathrm{trace}(\xi_X X^{-1} \eta_X X^{-1}). \tag{2.1.1}$$

The Karcher mean of $\{A_1, \ldots, A_K\}$, also called the Riemannian center of mass, is the minimizer of the sum of squared distances

$$\mu = \operatorname*{arg\,min}_{X \in \mathcal{S}_{++}^n} F(X), \quad \text{with } F : \mathcal{S}_{++}^{\mathrm{n}} \to \mathbb{R},\ X \mapsto \frac{1}{2K} \sum_{i=1}^{K} \delta^2(X, A_i), \tag{2.1.2}$$

where $\delta(p, q) = \| \log(p^{-1/2} q p^{-1/2}) \|_{\mathrm{F}}$ is the geodesic distance associated with Riemannian metric (2.1.1). It is proved in [56] that function F has a unique minimizer. Hence a point $\mu \in \mathcal{S}_{++}^n$ is a Karcher mean if it is a stationary point of F, i.e., $\operatorname{grad} F(\mu) = 0$, where $\operatorname{grad} F$ denotes the Riemannian gradient of F under metric (2.1.1). However, a closed-form solution for problem (2.1.2) is unknown in general, and for this reason, the Karcher mean is usually computed by iterative methods.

Related work. Various methods have been used to compute the Karcher mean of SPD matrices. Most of them resort to the framework of Riemannian optimization (see, e.g., [2]). In particular, [53] presents a survey of several optimization algorithms, including Riemannian versions of steepest descent, conjugate gradient, BFGS, and trust-region Newton methods. The authors conclude that the first order methods, steepest descent and conjugate gradient, are the preferred choices for problem (2.1.2) in terms of computation time. The benefit of fast convergence of Newton's method

and BFGS is nullified by their high computational costs per iteration, especially as the size of the matrices increases. It is also empirically observed in [53] that the Riemannian metric yields much faster convergence for their tested algorithms compared with the induced Euclidean metric, which is given by $g_X(\eta_X, \xi_X) = \mathrm{trace}(\xi_X \eta_X)$.

A Riemannian version of the Barzilai-Borwein method (RBB) has been considered in [52]. Several stepsize selection rules have been investigated for the Riemannian steepest descent (RSD) method. A constant stepsize strategy is proposed in [83] and a convergence analysis is given. An adaptive stepsize selection rule based on the explicit expression of the Riemannian Hessian of the cost function F is studied in [82, Algorithm 2], and is shown to be the optimal stepsize for strongly convex function in Euclidean space, see [73, Theorem 2.1.14]. That is, the stepsize is chosen as $\alpha_k = 2/(M_k+L_k)$, where M_k and L_k are the lower and upper bounds on the eigenvalues of the Riemannian Hessian of F, respectively. A version of Newton method for the Karcher mean computation is also provided in [82]. A Richardson-like iteration is derived and evaluated empirically in [16], and is available in the Matrix Means Toolbox[1]. It is seen in Section 1.4.5 that the Richardson-like iteration is a steepest descent method with stepsize $\alpha_k = 1/L_k$.

The main contributions of this dissertation for the SPD Karcher mean computation are:

- By providing lower and upper bounds on the condition number of the Riemannian and Euclidean Hessians of the cost function (2.1.2), we give a theoretical explanation for the above-mentioned behavior of the Riemannian and Euclidean steepest descent algorithms for SPD Karcher mean computation. Then we provide a detailed description of a limited-memory Riemannian BFGS (LRBFGS) method for this mean computation problem. Riemannian optimization methods such as LRBFGS involve manipulation of geometric objects on manifolds, such as tangent vectors, evaluation of a Riemannian metric, retraction, and vector transport. We present detailed methods to produce efficient numerical representations of those objects on the $\mathcal{S}^{\mathrm{n}}_{++}$ manifold. In fact, there are several alternatives to choose from for geometric objects on $\mathcal{S}^{\mathrm{n}}_{++}$. We offer theoretical and empirical suggestions on how to choose between those alternatives for LRBFGS based on computational complexity analysis and numerical experiments. Our numerical experiments indicate that as a result, and in spite of the favorable bound on the Riemannian Hessian that ensures Riemannian steepest descent to be an efficient method, the obtained LRBFGS method outperforms state-of-the-art methods on various instances of the problem. We also show that RBB is a special case of LRBFGS.

[1] `http://bezout.dm.unipi.it/software/mmtoolbox/`

- Another contribution of our work is to provide a C++ toolbox for the SPD Karcher mean computation, which includes LRBFGS, RBFGS, RBB, and RSD. The toolbox[2] relies on ROPTLIB, an object-oriented C++ library for optimization on Riemannian manifolds [50]. To the best of our knowledge, there is no other publicly available C++ toolbox for the SPD Karcher mean computation. Our previous work [96] provides a MATLAB implementation[3] for this problem. The Matrix Means Toolbox[1] developed by Bini et al. in [16] is also written in MATLAB. As an interpreted language, MATLAB's execution efficiency is lower than compiled languages, such as C++. In addition, the timing measurements in MATLAB can be skewed by MATLAB's overhead, especially for small-size problems. As a result, we resort to C++ for efficiency and reliable timing.

- We test the performance of LRBFGS on problems of various sizes and conditioning, and compare with the state-of-the-art methods mentioned above. The size of a problem is characterized by the number of matrices as well as the dimension of each matrix, and the conditioning of the problem is characterized by the condition number of matrices. It is shown empirically that LRBFGS is appropriate for large-size problems or ill-conditioned problems. Especially when one has little knowledge of the conditioning of a problem, LRBFGS becomes the method of choice since it is robust to problem conditioning and parameter setting. The numerical results also illustrate the speedup of using C++ vs. MATLAB, especially for small-size problems. It is observed that the C++ implementation is faster than MATLAB by a factor of 100 or more with the factor gradually reducing as the size of the problem gets larger.

2.2 Analysis on the conditioning of the problem

The convergence speed of optimization methods depends on the conditioning of the Hessian of the cost function at the minimizer. Large values of condition number lead to slow convergence of optimization algorithms, especially for steepest descent methods. The choice of the metric has an important influence on the difficulty of an optimization problem via influencing the conditioning of the Hessian of the cost function. A good choice of metric may reduce the condition number of the Hessian.

Rentmeesters et al. [82, inequality (3.29)] give bounds on the eigenvalues of the Riemannian Hessian of the squared distance function $f_A(X) = \frac{1}{2}\delta^2(X, A)$ given $A \in \mathcal{S}^{\mathrm{n}}_{++}$. On this basis, the bounds on the eigenvalues of the Riemannian Hessian of F can be obtained trivially. We summarize the results from [82] in Theorem 2.2.1 and, for completeness, we give the proof omitted by [82].

[2] `http://www.math.fsu.edu/~whuang2/papers/RMKMSPDM.htm`
[3] `http://www.math.fsu.edu/~whuang2/papers/ARLBACMGM.htm`

Theorem 2.2.1. *Let F be the objective function defined in problem (2.1.2) and $X \in \mathcal{S}^{\mathrm{n}}_{++}$. Then the eigenvalues of the Riemannian Hessian of F at X are bounded by*

$$1 \leq \frac{\operatorname{Hess} F(X)[\Delta X, \Delta X]}{\|\Delta X\|^2} \leq 1 + \frac{\log(\max_i \kappa_i)}{2}, \tag{2.2.1}$$

where κ_i denotes the condition number of matrix $X^{-1/2} A_i X^{-1/2}$ (or equivalently $L_x^{-1} A_i L_x^{-T}$ with $X = L_x L_x^T$ being the Cholesky decomposition of X).

Proof. The proof is a simple generalization from [82, inequality (3.29)], which gives bounds on the eigenvalues of the Riemannian Hessian of the function $f_A(X)$ as

$$1 \leq \frac{\operatorname{Hess} f_A(X)[\Delta X, \Delta X]}{\|\Delta X\|^2} \leq \frac{\log \kappa}{2} \coth(\frac{\log \kappa}{2}), \tag{2.2.2}$$

where κ is condition number of $X^{-1/2} A X^{-1/2}$. Notice that the objective function $F(X) = \frac{1}{K} \sum_{i=1}^{K} f_{A_i}(X)$. Thus, we have

$$1 \leq \frac{\operatorname{Hess} F(X)[\Delta X, \Delta X]}{\|\Delta X\|^2} \leq \frac{1}{K} \sum_{i=1}^{K} \frac{\log \kappa_i}{2} \coth(\frac{\log \kappa_i}{2}). \tag{2.2.3}$$

Since $x \coth(x)$ is strictly increasing and bounded by $1 + x$ on $[0, \infty]$, the right hand side of inequality (2.2.3) is thus bounded by $1 + \log(\max_i \kappa_i)/2$. □ □

Theorem 2.2.1 implies that we should not expect a very ill-conditioned Riemannian Hessian in practice. However, this is not the case when the Euclidean metric is used. In Theorem 2.2.2, we derive bounds on the condition number of the Euclidean Hessian of F at the minimizer. We need the following lemma before deriving the bounds.

Lemma 2.2.1. *Let $A \in \mathcal{S}^{\mathrm{n}}_{++}$ be a symmetric positive definite matrix with eigenvalues satisfying $0 < \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$, and $\eta \in \mathbb{R}^{n \times n}$ be an $n \times n$ real symmetric matrix. Then, we have*

$$\max_{\eta = \eta^T} \frac{\operatorname{tr}(A\eta A\eta)}{\|\eta\|_{\mathrm{F}}^2} = \lambda_n^2, \ \textit{and} \ \min_{\eta = \eta^T} \frac{\operatorname{tr}(A\eta A\eta)}{\|\eta\|_{\mathrm{F}}^2} = \lambda_1^2. \tag{2.2.4}$$

Proof. Let $A = Q \Sigma Q^T$ be the eigenvalues decomposition of A, where $QQ^T = I$ and $\Sigma = \operatorname{diag}(\lambda_1, \ldots, \lambda_n)$. Then, we have

$$\frac{\operatorname{tr}(A\eta A\eta)}{\|\eta\|_{\mathrm{F}}^2} = \frac{\operatorname{tr}(Q\Sigma Q^T \eta^T Q \Sigma Q^T \eta)}{\|Q^T \eta Q\|_{\mathrm{F}}^2} = \frac{\operatorname{tr}(\Sigma \tilde{\eta}^T \Sigma \tilde{\eta})}{\|\tilde{\eta}\|_{\mathrm{F}}^2}. \tag{2.2.5}$$

Notice that we can rewrite the trace term on the right hand of equation (2.2.5) as

$$\operatorname{tr}(\Sigma\tilde{\eta}^T\Sigma\tilde{\eta}) = \operatorname{vec}(\tilde{\eta}\Sigma)^T \operatorname{vec}(\Sigma\eta) = \operatorname{vec}(\tilde{\eta})^T(I_n\otimes\Sigma)(\Sigma\otimes I_n)\operatorname{vec}(\tilde{\eta}) = \operatorname{vec}(\tilde{\eta})^T(\Sigma\otimes\Sigma)\operatorname{vec}(\tilde{\eta}). \quad (2.2.6)$$

On one hand, we have

$$\max_{\tilde{\eta}=\tilde{\eta}^T} \frac{\operatorname{tr}(\Sigma\tilde{\eta}^T\Sigma\tilde{\eta})}{\|\tilde{\eta}\|_{\mathrm{F}}^2} \leq \max_{\tilde{\eta}\in\mathbb{R}^{n\times n}} \frac{\operatorname{tr}(\Sigma\tilde{\eta}^T\Sigma\tilde{\eta})}{\|\tilde{\eta}\|_{\mathrm{F}}^2} = \max_{\tilde{\eta}\in\mathbb{R}^{n\times n}} \frac{\operatorname{vec}(\tilde{\eta})^T(\Sigma\otimes\Sigma)\operatorname{vec}(\tilde{\eta})}{\operatorname{vec}(\tilde{\eta})^T\operatorname{vec}(\tilde{\eta})} = \lambda_n^2. \quad (2.2.7)$$

The last equality comes from the fact that the eigenvalues of $\Sigma\otimes\Sigma$ are $\lambda_i\lambda_j$, $i = 1,\ldots,n$, $j = 1,\ldots,n$. On the other hand,

$$\max_{\tilde{\eta}=\tilde{\eta}^T} \frac{\operatorname{tr}(\Sigma\tilde{\eta}^T\Sigma\tilde{\eta})}{\|\tilde{\eta}\|_{\mathrm{F}}^2} \geq \frac{\operatorname{tr}(\Sigma\eta_0^T\Sigma\eta_0)}{\|\eta_0\|_{\mathrm{F}}^2} = \lambda_n^2, \quad (2.2.8)$$

where $\eta_0 = e_n e_n^T$ and $e_n = (0,\ldots,0,1)^T$. That is, η_0 is an $n\times n$ zero matrix except the (n,n) entry is 1. Combining inequalities (2.2.7) and (2.2.8), we obtain the first part of (2.2.4).

Similarly, we have

$$\min_{\tilde{\eta}=\tilde{\eta}^T} \frac{\operatorname{tr}(\Sigma\tilde{\eta}^T\Sigma\tilde{\eta})}{\|\tilde{\eta}\|_{\mathrm{F}}^2} \geq \min_{\tilde{\eta}\in\mathbb{R}^{n\times n}} \frac{\operatorname{tr}(\Sigma\tilde{\eta}^T\Sigma\tilde{\eta})}{\|\tilde{\eta}\|_{\mathrm{F}}^2} = \min_{\tilde{\eta}\in\mathbb{R}^{n\times n}} \frac{\operatorname{vec}(\tilde{\eta})^T(\Sigma\otimes\Sigma)\operatorname{vec}(\tilde{\eta})}{\operatorname{vec}(\tilde{\eta})^T\operatorname{vec}(\tilde{\eta})} = \lambda_1^2. \quad (2.2.9)$$

On the other hand, we let $\eta_0 = e_1 e_1^T$ and $e_1 = (1,0,\ldots,0)^T$. Then, we have

$$\min_{\tilde{\eta}=\tilde{\eta}^T} \frac{\operatorname{tr}(\Sigma\tilde{\eta}^T\Sigma\tilde{\eta})}{\|\tilde{\eta}\|_{\mathrm{F}}^2} \leq \frac{\operatorname{tr}(\Sigma\eta_0^T\Sigma\eta_0)}{\|\eta_0\|_{\mathrm{F}}^2} = \lambda_1^2. \quad (2.2.10)$$

Combining inequalities (2.2.9) and (2.2.10) immediately lead to the second part of (2.2.4). □ □

For notational simplicity, we use super script 'E' and 'R' to differentiate the Euclidean metric and the Riemannian metric.

Theorem 2.2.2. *Let $f : \mathcal{S}_{++}^{\mathrm{n}} \to \mathbb{R}$ be twice continuously differentiable and μ be a stationary point for f. Assume the largest and smallest eigenvalues of the Riemannian Hessian of f at μ are $\Lambda_{\max}$ and $\Lambda_{\min}$ respectively, i.e.,*

$$\Lambda_{\min} \leq \frac{\langle \operatorname{Hess}^{\mathrm{R}} f(\mu)[\eta],\eta\rangle^{\mathrm{R}}}{\langle\eta,\eta\rangle^{\mathrm{R}}} \leq \Lambda_{\max}. \quad (2.2.11)$$

Then the condition number of the Euclidean Hessian of f at μ, denoted by $\kappa(H^{\mathrm{E}})$, is bounded by

$$\frac{1}{\kappa(H^{\mathrm{R}})}\kappa^2(\mu) \leq \kappa(H^{\mathrm{E}}) \leq \kappa(H^{\mathrm{R}})\kappa^2(\mu), \quad (2.2.12)$$

where $\kappa(\mu)$ is the condition number of μ, and $\kappa(H^{\mathrm{R}}) = \Lambda_{\max}/\Lambda_{\min}$ is the condition number of the Riemannian Hessian of f at μ.

Proof. Recall that the condition number of the Euclidean Hessian of f at μ can be expressed as

$$\kappa(H^{\mathrm{E}}) = \max_{\eta=\eta^T} \frac{\langle \mathrm{Hess}^{\mathrm{E}} f(\mu)[\eta], \eta\rangle^{\mathrm{E}}}{\langle \eta, \eta\rangle^{\mathrm{E}}} / \min_{\eta=\eta^T} \frac{\langle \mathrm{Hess}^{\mathrm{E}} f(\mu)[\eta], \eta\rangle^{\mathrm{E}}}{\langle \eta, \eta\rangle^{\mathrm{E}}}, \tag{2.2.13}$$

and that of the Riemannian Hessian can be written in a similar way.

For any $X \in \mathcal{S}^{\mathrm{n}}_{++}$ and $\eta \in \mathrm{T}_X \mathcal{S}^{\mathrm{n}}_{++}$, the action of Hessian of f on η under the Riemannian metric is given by [53]

$$\mathrm{Hess}^{\mathrm{R}} f(X)[\eta] = \mathrm{D}(\mathrm{grad}^{\mathrm{R}} f)(X)[\eta] - \frac{1}{2}(\eta X^{-1} \mathrm{grad}^{\mathrm{R}} f(X) + \mathrm{grad}^{\mathrm{R}} f(X) X^{-1} \eta). \tag{2.2.14}$$

When $X = \mu$ is a stationary point of f, i.e., $\mathrm{grad}^{\mathrm{R}} f(\mu) = 0$, we have $\mathrm{Hess}^{\mathrm{R}} f(\mu)[\eta] = \mathrm{D}(\mathrm{grad}^{\mathrm{R}} f)(\mu)[\eta]$. As $\langle \mathrm{grad}^{\mathrm{E}} f(X), \eta\rangle^{\mathrm{E}} = \mathrm{D} f(X)[\eta] = \langle \mathrm{grad}^{\mathrm{R}} f(X), \eta\rangle^{\mathrm{R}}$, we have

$$\mathrm{grad}^{\mathrm{R}} f(X) = X \,\mathrm{grad}^{\mathrm{E}} f(X) X. \tag{2.2.15}$$

Therefore, equation (2.2.14) yields

$$\mathrm{Hess}^{\mathrm{R}} f(\mu)[\eta] = \eta \,\mathrm{grad}^{\mathrm{E}} f(\mu)\mu + \mu(\mathrm{D}(\mathrm{grad}^{\mathrm{E}} f)(\mu)[\eta])\mu + \mu \,\mathrm{grad}^{\mathrm{E}} f(\mu)\eta \tag{2.2.16}$$

$$= \mu(\mathrm{D}(\mathrm{grad}^{\mathrm{E}} f)(\mu)[\eta])\mu = \mu(\mathrm{Hess}^{\mathrm{E}} f(\mu)[\eta])\mu. \tag{2.2.17}$$

This gives us

$$\langle \mathrm{Hess}^{\mathrm{R}} f(\mu)[\eta], \eta\rangle^{\mathrm{R}} = \mathrm{tr}(\mu^{-1} \mathrm{Hess}^{\mathrm{R}} f(\mu)[\eta] \mu^{-1} \eta) = \mathrm{tr}(\mathrm{Hess}^{\mathrm{E}} f(\mu)[\eta]\eta) = \langle \mathrm{Hess}^{\mathrm{E}} f(\mu)[\eta], \eta\rangle^{\mathrm{E}}. \tag{2.2.18}$$

It follows that

$$\frac{\langle \mathrm{Hess}^{\mathrm{E}} f(\mu)[\eta], \eta\rangle^{\mathrm{E}}}{\langle \eta, \eta\rangle^{\mathrm{E}}} = \frac{\langle \mathrm{Hess}^{\mathrm{R}} f(\mu)[\eta], \eta\rangle^{\mathrm{R}}}{\langle \eta, \eta\rangle^{\mathrm{R}}} \cdot \frac{\langle \eta, \eta\rangle^{\mathrm{R}}}{\langle \eta, \eta\rangle^{\mathrm{E}}}. \tag{2.2.19}$$

By assumption, we have

$$\Lambda_{\min} \leq \frac{\langle \mathrm{Hess}^{\mathrm{R}} f(\mu)[\eta], \eta\rangle^{\mathrm{R}}}{\langle \eta, \eta\rangle^{\mathrm{R}}} \leq \Lambda_{\max}. \tag{2.2.20}$$

Assume that the eigenvalues of μ are $0 < \lambda_1 \leq \cdots \leq \lambda_n$, and then the eigenvalues of μ^{-1} are $0 < 1/\lambda_n \leq \cdots \leq 1/\lambda_1$. From Lemma 2.2.1, we have

$$\max_{\eta=\eta^T} \frac{\langle \eta, \eta\rangle^{\mathrm{R}}}{\langle \eta, \eta\rangle^{\mathrm{E}}} = \frac{1}{\lambda_1^2} \text{ and } \min_{\eta=\eta^T} \frac{\langle \eta, \eta\rangle^{\mathrm{R}}}{\langle \eta, \eta\rangle^{\mathrm{E}}} = \frac{1}{\lambda_n^2}. \tag{2.2.21}$$

Multiplying inequality (2.2.20) and equation (2.2.21) gives

$$\frac{\Lambda_{\min}}{\lambda_1^2} \leq \max_{\eta=\eta^T} \frac{\langle \mathrm{Hess}^{\mathrm{E}} f(\mu)[\eta], \eta\rangle^{\mathrm{E}}}{\langle \eta, \eta\rangle^{\mathrm{E}}} \leq \frac{\Lambda_{\max}}{\lambda_1^2}, \tag{2.2.22}$$

and

$$\frac{\Lambda_{\min}}{\lambda_n^2} \leq \min_{\eta=\eta^T} \frac{\langle \mathrm{Hess}^{\mathrm{E}} f(\mu)[\eta], \eta\rangle^{\mathrm{E}}}{\langle \eta, \eta\rangle^{\mathrm{E}}} \leq \frac{\Lambda_{\max}}{\lambda_n^2}. \tag{2.2.23}$$

Dividing inequality (2.2.22) by (2.2.23) gives us the lower and upper bounds on the condition number of the Euclidean Hessian of f at stationary point μ:

$$\frac{\Lambda_{\min}}{\Lambda_{\max}} \frac{\lambda_n^2}{\lambda_1^2} \leq \kappa(H^{\mathrm{E}}) \leq \frac{\Lambda_{\max}}{\Lambda_{\min}} \frac{\lambda_n^2}{\lambda_1^2}. \tag{2.2.24}$$

That is,

$$\frac{1}{\kappa(H^{\mathrm{R}})} \kappa^2(\mu) \leq \kappa(H^{\mathrm{E}}) \leq \kappa(H^{\mathrm{R}}) \kappa^2(\mu). \tag{2.2.25}$$

□ □

Remark 2.2.1. *Notice that equality* (2.2.18) *can be simply obtained by the fact that* $\langle \mathrm{Hess}\, F(\mu)[\eta], \eta\rangle$ *is independent of the metric when* μ *is a critical point of* F*. The proof can be found in [2, Section 5.5].*

For the cost function F in (2.1.2), it is seen from Theorem 2.2.2 that the condition number of the Euclidean Hessian at the minimizer (stationary point) is bounded below by the square of the condition number of the minimizer scaled by the reciprocal of the condition number of the Riemannian Hessian of F. Hence when the minimizer is ill-conditioned, the Euclidean Hessian of F at the minimizer is ill-conditioned as well, which will slow down the optimization methods. Our numerical experiments in Section in 2.5.3 demonstrate this expectation.

2.3 Implementation techniques on $\mathcal{S}_{++}^n$

This section is devoted to the implementation details of the required objects for Riemannian optimization methods on $\mathcal{S}_{++}^{\mathrm{n}}$, as well as the SPD Karcher mean computation problem. Manifold-related objects include tangent vectors, the Riemannian metric, isometric vector transport, and retraction. Problem-related objects include the cost function and Riemannian gradient evaluations. We also provide a floating point operation (flop) count[4], for most operations.

[4] see [40, Section 1.2.4]

2.3.1 Representations of a tangent vector and Riemannian metric on $\mathcal{S}_{++}^n$

As mentioned in Section 1.5, the tangent space of $\mathcal{S}_{++}^{\mathrm{n}}$ at X is the set of symmetric matrices, i.e., $\mathrm{T}_X\,\mathcal{S}_{++}^n = \{S \in \mathbb{R}^{n\times n} | S = S^T\}$. The dimension of $\mathcal{S}_{++}^{\mathrm{n}}$ is $d = n(n+1)/2$. Thus, a tangent vector η_X in $\mathrm{T}_X\,\mathcal{S}_{++}^n$ can be represented either by an n^2-dimensional vector in Euclidean space $\mathcal{E}$, or a d-dimensional vector of coordinates in a given basis B_X of $\mathrm{T}_X\,\mathcal{S}_{++}^n$. The n^2-dimensional representation is called the extrinsic approach, and the d-dimensional one is called the intrinsic approach. For notational simplicity, we use w to denote the dimension of the embedding space, and d to denote the dimension of the manifold, i.e., $w = n^2$ and $d = n(n+1)/2$.

The computational benefits of using intrinsic representation are addressed in [47,48]: (i) Working in d-dimension reduces the computational complexity of linear operations on the tangent space. (ii) There exists an isometric vector transport, called vector transport by parallelization, whose intrinsic implementation is simply the identity. (iii) The Riemannian metric can be reduced to the Euclidean metric. However, the intrinsic representation requires a basis of tangent space, and in order to obtain the computational benefits mentioned above, it must be orthonormal. Hence, if a manifold admits a smooth field of orthonormal tangent space bases with acceptable computational complexity, the intrinsic representation often leads to a very efficient implementation. This property holds for $\mathcal{S}_{++}^n$ as shown next.

The orthonormal basis B_X of $\mathrm{T}_X\,\mathcal{S}_{++}^n$ that we select is given by

$$\{Le_ie_i^TL^T : i = 1,\ldots,n\}\bigcup\{\frac{1}{\sqrt{2}}L(e_ie_j^T + e_je_i^T)L^T,\ i<j,\ i=1,\ldots,n,\ j=1,\ldots,n\}, \qquad (2.3.1)$$

where $X = LL^T$ denotes the Cholesky decomposition, and $\{e_1,\ldots,e_n\}$ is the standard basis of n-dimensional Euclidean space. Another choice is to use the matrix square root $X^{1/2}$ instead of Cholesky decomposition of X, which however costs more [44]. It is easy to verify the orthonormality of B_X under the Riemannian metric (2.1.1), i.e., $B_X^\flat B_X = I_{d\times d}$ for all $X \in \mathcal{S}_{++}^{\mathrm{n}}$. We assume throughout this section that B_X stands for our selected orthonormal basis of $\mathrm{T}_X\,\mathcal{S}_{++}^n$ defined in (2.3.1).

Notice that $\{e_ie_i^T : i = 1,\ldots,n\}\bigcup\{\frac{1}{\sqrt{2}}(e_ie_j^T + e_je_i^T),\ i<j,\ i=1,\ldots,n,\ j=1,\ldots,n\}$ also forms a basis for $\mathrm{T}_X\,\mathcal{S}_{++}^{\mathrm{n}}$. However, this basis is orthonormal under the Euclidean metric (1.5.7), not the affine-invariant metric (2.1.1).

Let η_X be a tangent vector in $\mathrm{T}_X\,\mathcal{S}_{++}^{\mathrm{n}}$ and v_X be its intrinsic representation. We define function $E2D_X : \eta_X \mapsto v_X = B_X^\flat\eta_X$ that maps the extrinsic representation to the intrinsic representation.

Using the orthonormal basis defined in (2.3.1), the intrinsic representation of η_X is obtained by taking the diagonal elements of $L^{-1}\eta_X L^{-T}$, and its upper triangular elements row-wise and multiplied by $\sqrt{2}$. A detailed description of function $E2D$ is given in Algorithm 1. The number of flops for each step is given on the right of the algorithm.

Since B_X forms an orthonormal basis of $\mathrm{T}_X\,\mathcal{S}^{\mathrm{n}}_{++}$, the Riemannian metric (2.1.1) reduces to the Euclidean metric under the intrinsic representation, i.e.,

$$\tilde{g}_X(v_X, u_X) := g_X(\eta_X, \xi_X) = g_X(B_X v_X, B_X u_X) = v_X^T u_X, \tag{2.3.2}$$

where $\eta_X = B_X v_X$, $\xi_X = B_X u_X \in \mathrm{T}_X\,\mathcal{S}^{\mathrm{n}}_{++}$. The evaluation of (2.3.2) requires $2d$ flops, which is cheaper than the evaluation of (2.1.1).

For the intrinsic approach, retractions require mapping the intrinsic representation of tangent vectors back to the extrinsic representation, which may need extra work. Let function $D2E_X : v_X \mapsto \eta_X = B_X v_X$ denote this mapping. In practice, the function $D2E_X$ using basis (2.3.1) is described in Algorithm 2.

Algorithm 1 Compute $E2D_X(\eta_X)$

Input: $X = LL^T \in \mathcal{S}^{\mathrm{n}}_{++}$, $\eta_x \in \mathrm{T}_X\,\mathcal{S}^{\mathrm{n}}_{++}$.

1: Compute $Y = L^{-1}\eta_X$ by solving linear system $LY = \eta_X$; ▷ # n^3

2: $Y \leftarrow Y^T$ (i.e., $Y = \eta_X L^{-T}$) ;

3: Compute $Z = L^{-1}\eta_X L^{-T}$ by solving linear system $LZ = Y$; ▷ # n^3

4: return $v_X = (z_{11}, \ldots, z_{nn}, \sqrt{2}z_{12}, \ldots, \sqrt{2}z_{1n}, \sqrt{2}z_{23}, \ldots, \sqrt{2}z_{2n}, \ldots, \sqrt{2}z_{(n-1)n})^T$; ▷ # d

2.3.2 Retraction and vector transport on $\mathcal{S}^n_{++}$

The choice of retraction and vector transport is a key step in the design of efficient Riemannian optimization algorithms. As seen in Section 1.4.4, the exponential mapping is a natural choice for retraction. When $\mathcal{S}^n_{++}$ is endowed with the Riemannian metric (2.1.1), the exponential mapping is given by, see [35],

$$\mathrm{Exp}_X(\eta_X) = X^{1/2}\exp(X^{-1/2}\eta_X X^{-1/2})X^{1/2}, \tag{2.3.3}$$

for all $X \in \mathcal{S}^n_{++}$ and $\eta_X \in \mathrm{T}_X\,\mathcal{S}^n_{++}$. Using the Cholesky factorization instead of the matrix square root, (2.3.3) can be rewritten as

$$\mathrm{Exp}_X(\eta_X) = L\exp(L^{-1}\eta_X L^{-T})L^T, \tag{2.3.4}$$

Algorithm 2 Compute $D2E_X(v_X)$

Input: $X = LL^T \in \mathcal{S}^{\mathrm{n}}_{++}$, $v_x \in \mathbb{R}^{n(n+1)/2}$.

1: **for** $i = 1, \ldots, n$ **do** ▷ # n
2: $\quad \eta_{ii} = v_X(i)$;
3: **end for**
4: $k = n + 1$;
5: **for** $i = 1, \ldots, n$ **do** ▷ # $n^2 - n(n+1)/2$
6: $\quad$ **for** $j = i + 1, \ldots, n$ **do**
7: $\quad\quad \eta_{ij} = v_X(k)$ and $\eta_{ji} = v_X(k)$;
8: $\quad\quad k = k + 1$;
9: $\quad$ **end for**
10: **end for**
11: return $L\eta L^T$; ▷ # $2n^3$

where $X = LL^T$. In practice, the exponential mapping (2.3.4) is expensive to compute due to the matrix exponential. The exponential of matrix M is computed as $\exp(M) = U \exp(\Sigma) U^T$, with $M = U\Sigma U^T$ being the eigenvalue decomposition. Obtaining Σ and U by Golub-Reinsch algorithm requires $12n^3$ flops, see [40, Figure 8.6.1]. Hence the evaluation of $\exp(M)$ requires $16n^3$ flops in total. More importantly, when computing the matrix exponential $\exp(M)$, eigenvalues of large magnitude can lead to numerical difficulties. Jeuris et al. [53] proposed a retraction

$$R_X(\eta_X) = X + \eta_X + \frac{1}{2}\eta_X X^{-1} \eta_X, \tag{2.3.5}$$

which is a second order approximation to the exponential mapping (2.3.4). Retraction (2.3.5) is cheaper to compute and requires $3n^3 + o(n^3)$ flops. More importantly, it tends to avoid numerical overflow. An important property of retraction (2.3.5) is stated in Proposition 2.3.1.

Proposition 2.3.1. *Retraction $R_X(\eta)$ defined in* (2.3.5) *remains symmetric positive-definite for all $X \in \mathcal{S}^n_{++}$ and $\eta \in \mathrm{T}_X \mathcal{S}^{\mathrm{n}}_{++}$.*

Proof. For all $X \in \mathcal{S}^{\mathrm{n}}_{++}$ and $\eta \in \mathrm{T}_X \mathcal{S}^{\mathrm{n}}_{++}$, we have

$$\begin{aligned} X + \eta + \frac{1}{2}\eta X^{-1} \xi &= \frac{1}{2}(X + 2\eta + \eta X^{-1} \eta) + \frac{1}{2} X \\ &= \frac{1}{2}(X^{1/2} + \eta X^{-1/2})(X^{1/2} + \eta X^{-1/2})^T + \frac{1}{2} X \end{aligned} \tag{2.3.6}$$

Then, for any $v \neq 0$,

$$\begin{aligned} v^T R_X(\eta) v &= \frac{1}{2} v^T (X^{1/2} + \eta X^{-1/2})(X^{1/2} + \eta X^{-1/2})^T v + \frac{1}{2} v^T X v \\ &= \frac{1}{2} \{(X^{1/2} + \eta X^{-1/2})^T v\}^T \{(X^{1/2} + \eta X^{-1/2})^T v\} + \frac{1}{2} v^T X v \\ &> 0 \end{aligned} \tag{2.3.7}$$

□

Another retraction that can be computed efficiently is the first order approximation to (2.3.4), i.e.,

$$R_X(\eta_X) = X + \eta_X. \tag{2.3.8}$$

In fact, retraction (2.3.8) is the exponential mapping when $\mathcal{S}^{\mathrm{n}}_{++}$ is endowed with the Euclidean metric (1.5.7). However, the result of retraction (2.3.8) is not guaranteed to be positive definite. Therefore one must be careful when using this Euclidean retraction. One remedy is to reduce the stepsize when necessary. The Richardson-like iteration in [16] is a steepest descent method using Euclidean retraction (2.3.8).

Parallel translation is a particular instance of vector transport. The parallel translation on $\mathcal{S}^n_{++}$ is given by, see [35],

$$\mathcal{T}_{p_{\xi_X}}(\eta_X) = X^{1/2} \exp(\frac{X^{-1/2} \xi_X X^{-1/2}}{2}) X^{-1/2} \eta_X X^{-1/2} \exp(\frac{X^{-1/2} \xi_X X^{-1/2}}{2}) X^{1/2}. \tag{2.3.9}$$

Again, we can rewrite parallel translation (2.3.9) as

$$\mathcal{T}_{p_{\xi_X}}(\eta_X) = L \exp(\frac{L^{-1} \xi_X L^{-T}}{2}) L^{-1} \eta_X L^{-T} \exp(\frac{L^{-1} \xi_X L^{-T}}{2}) L^T, \tag{2.3.10}$$

where $X = LL^T$. The computation of parallel translation involves the matrix exponential, which is computationally expensive. Note, however, that if parallel translation is used together with the exponential mapping (2.3.4), the most expensive exponential computation can be shared by rewriting (2.3.10) and (2.3.4) as shown in Algorithm 4. Even so, the matrix exponential computation is still required. We thus resort to another vector transport.

Recently, Huang et al. [47, Section 2.3.1] proposed a novel way to construct an isometric vector transport, called vector transport by parallelization. It is defined by

$$\mathcal{T}_S = B_Y B_X^\flat, \tag{2.3.11}$$

where B_X and B_Y are orthonormal bases of $\mathrm{T}_X\,\mathcal{S}_{++}^{\mathrm{n}}$ and $\mathrm{T}_Y\,\mathcal{S}_{++}^{\mathrm{n}}$ defined in (2.3.1) respectively. Let $v_X = B_X^\flat \eta_X$ be the intrinsic representation of η_X. Then, the intrinsic approach of (2.3.11), denoted by $\mathcal{T}_S^d$, is given by

$$\mathcal{T}_S^d v_X = B_Y^\flat \mathcal{T}_S \eta_X = B_Y^\flat B_Y B_X^\flat B_X v_X = v_X \tag{2.3.12}$$

That is, the intrinsic representation of vector transport by parallelization is simply the identity, which is the cheapest vector transport one can expect.

Another possible choice for the vector transport is the identity mapping: $\mathcal{T}_{id_{\xi_X}}(\eta_X) = \eta_X$. However, vector transport $\mathcal{T}_{id}$ is not applicable to the LRBFGS in [51, Algorithm 2] since it is not isometric under Riemannian metric (2.1.1).

Given a retraction, Huang et al. [51, Section 2] provides a method to construct an isometric vector transport such that the pair satisfies the locking condition[5], denoted by $\mathcal{T}_L$, which is given by

$$\mathcal{T}_{L_{\xi_X}} \eta_X = B_Y (I - \frac{2v_2 v_2^T}{v_2^T v_2})(I - \frac{2v_1 v_1^T}{v_1^T v_1}) B_1^\flat \eta_X, \tag{2.3.13}$$

where $v_1 = B_X^\flat \xi_X - z$, $v_2 = z - \beta B_Y^\flat \mathcal{T}_{R_{\xi_X}} \xi_X$, $\beta = \|\xi_X\|/\|\mathcal{T}_{R_{\xi_X}} \xi_X\|$, and $\mathcal{T}_R$ denotes the differentiated retraction. z can be any tangent vector satisfying $\|z\| = \|B_1^\flat \xi_X\| = \|\beta B_2^\flat \mathcal{T}_{R_{\xi_X}} \xi_X\|$. In particular, $z = -B_1^\flat \xi_X$ and $z = -\beta B_2^\flat \xi_X$ are natural choices. The intrinsic representation of (2.3.13) is given by

$$\mathcal{T}_L^d v_X = B_Y^\flat B_Y (I - \frac{2v_2 v_2^T}{v_2^T v_2})(I - \frac{2v_1 v_1^T}{v_1^T v_1}) B_1^\flat \eta_X \tag{2.3.14}$$

$$= (I - \frac{2v_2 v_2^T}{v_2^T v_2})(I - \frac{2v_1 v_1^T}{v_1^T v_1}) v_X. \tag{2.3.15}$$

Note that when computing (2.3.15), $(I - \frac{2v_1 v_1^T}{v_1^T v_1}) v_X$ is computed as $v_X - \frac{2(v_1^T v_X)}{v_1^T v_1} v_1$, which requires $6d$ flops. Hence the evaluation of intrinsic vector transport (2.3.15) requires $12d$ flops, i.e., $6n^2$.

Suppose retraction (2.3.5) is used, its differential is given by

$$\begin{aligned} \mathcal{T}_{R_{\eta_x}} \xi_x &= \mathrm{D}\, R_x(\eta_x)[\xi_x] \\ &= \xi_X + \frac{1}{2}(\eta_X X^{-1} \xi_X + \xi_X X^{-1} \eta_X). \end{aligned} \tag{2.3.16}$$

[5] see [51, Section 2 Equation (2.8)] for the definition of locking condition

2.3.3 Riemannian gradient of the sum of squared distances function

The cost function (2.1.2) can be rewritten as

$$f(X) = \frac{1}{2K}\sum_{i=1}^{K} \|\log(A_i^{-1/2} X A_i^{-1/2})\|_F^2 \tag{2.3.17}$$

$$= \frac{1}{2K}\sum_{i=1}^{K} \|\log(L_{A_i}^{-1} X L_{A_i}^{-T})\|_F^2 \tag{2.3.18}$$

where $A_i = L_{A_i} L_{A_i}^T$. We use Cholesky factorization rather than the matrix square root due to computational efficiency. The matrix logarithm is computed in a similar way as exponential, i.e., $\log(M) = U \log(\Sigma) U^T$ with $M = U\Sigma U^T$ being the eigenvalue decomposition. So the number of flops required by the evaluation of (2.3.18) is $18Kn^3$.

The Riemannian gradient of the cost function f in (2.1.2) is given by, see [56],

$$\operatorname{grad} f(X) = -\frac{1}{K}\sum_{i=1}^{K} \operatorname{Exp}_X^{-1}(A_i), \tag{2.3.19}$$

where $\operatorname{Exp}_X^{-1}(Y)$ is the log-mapping. On $\mathcal{S}_{++}^n$, the log-mapping is

$$\operatorname{Exp}_X^{-1}(Y) = X^{1/2} \log(X^{-1/2} Y X^{-1/2}) X^{1/2} = \log(Y X^{-1}) X. \tag{2.3.20}$$

Note that the computational complexity of the Riemannian gradient is less than that conveyed in formula (2.3.20) since the most expensive logarithm computation is already available from the evaluation of the cost function at X. Specifically, each term in (2.3.19) is computed as $-\operatorname{Exp}_X^{-1}(A_i) = -\log(A_i X^{-1}) X = \log(X A_i^{-1}) X = L_{A_i} \log(L_{A_i}^{-1} X L_{A_i}^{-T}) L_{A_i}^{-1} X$, and the term $\log(L_{A_i}^{-1} X L_{A_i}^{-T})$ is available from the evaluation of the cost function $F(X)$ in (2.3.18). Hence the computation of gradient requires $5Kn^3$ flops if $\log(L_{A_i}^{-1} X L_{A_i}^{-T})$ is given.

2.4 Description of the SPD Karcher mean computation methods

In this section, we discuss the algorithms for SPD Karcher mean computation. We first review the Riemannian steepest descent methods (RSD) without line search in the literature, including RSD with stepsize selection rule proposed by Q. Rentmeesters (RSD-QR) [82] and a Richardson-like iteration (RL) [16]. Then we review the line-search-based Riemannian Barzilai-Borwein (RBB) in [52]. Note that RBB is a steepest descent method combined with the Barzilai-Borwein stepsize.

We also consider an adaptive BB stepsize selection rule proposed in [39]. In particular, we applied the intrinsic approach to the implementation of RBB, and compare with the implementation in [52]. It is seen that the former implementation leads to a significant decrease in computation time. In the end we provide detailed descriptions of a line-search-based limited-memory Riemannian BFGS (LRBFGS) [51] on $\mathcal{S}_{++}^{\mathrm{n}}$.

All the algorithms considered are retraction-based methods, namely, an iterate x_k on a manifold $\mathcal{M}$ is updated by

$$x_{k+1} = R_{x_k}(\alpha_k \eta_k), \tag{2.4.1}$$

where R is a retraction on $\mathcal{M}$, $\eta_k \in \mathrm{T}_{x_k}\mathcal{M}$ is the search direction and $\alpha_k \in \mathbb{R}$ denotes the stepsize.

2.4.1 Riemannian steepest descent methods without line search

For the steepest descent method, the search direction in (2.4.1) is taken as the negative gradient, i.e.,

$$\eta_k = -\operatorname{grad} f(x_k). \tag{2.4.2}$$

We summarize RSD without line search for the SPD Karcher mean computation in Algorithm 3. The differences between RSD-QR and RL are the choices of stepsize strategy in Step 11 and retraction in Step 13 of Algorithm 3. For RL, the stepsize is taken as $\alpha_{RL} = 1/\Delta$, where Δ is the upper bound on the eigenvalues of the Hessian of the cost function as computed in Step 10, and the Euclidean retraction (2.3.8) is used. The number of flops required per iteration is $22Kn^3 + o(Kn^3)$. For RSD-QR, the chosen stepsize is $\alpha_{QR} = 2/(U + \Delta)$, where $U = 1$ is the lower bound on the eigenvalues of the Hessian of the cost function. It is easy to verify that $1/\Delta \leq 2/(1 + \Delta)$, with equality when $\Delta = 1$. Since the eigenvalues of the Hessian of the cost function are bounded by U and Δ, then $\Delta = 1$ implies that all the eigenvalues of the Hessian are exactly 1. So $\alpha_{RL} = \alpha_{QR}$ if and only if the Hessian of the cost function is the identity matrix, and we have $\alpha_{RL} < \alpha_{QR}$ in general. The exponential mapping (2.3.3) is used by RSD-QR in [82, Algorithm 2]. In practice, we use retraction (2.3.5), since the exponential mapping contains matrix exponential evaluation, and it turns out to be a problem if the eigenvalues of matrices in some intermediate iterations become too large, resulting in numerical overflow. Then each iteration in RSD-QR needs $22Kn^3 + 4/3n^3 + o(Kn^3)$ flops. Even though RSD-QR is slightly more expensive per iteration than RL due to the choice of retraction, it will be seen in our experiments to require fewer iterations to achieve

a desired tolerance, to perform very well on small-size problems in terms of computation time, and to consistently outperform RL in various situations.

Algorithm 3 RSD without line search for the SPD Karcher mean computation

Input: $A_i = L_{A_i} L_{A_i}^T$; tolerance for stopping criteria ϵ; initial iterate $x_0 \in \mathcal{S}_{++}^{\mathrm{n}}$;

$k = 0$;
while $\| \operatorname{grad} f(x_k) \| > \epsilon$ **do**
  for $i = 1, \ldots, K$ **do**
    Compute $M_i = L_{A_i}^{-1} x_k L_{A_i}^{-T}$; ▷ # $2n^3$
    Compute $M_i = U \Sigma U^{-1}$ and set $\lambda = \operatorname{diag}(\Sigma)$; ▷ # $12n^3$
    Compute the condition number $c_i = \max(\lambda) / \min(\lambda)$; ▷ # 1
    Compute $K_i = U \log(\Sigma) U^{-1}$; ▷ # $4n^3$
    Compute $G_i = L_{A_i} K_i L_{A_i}^{-1} x_k$; ▷ # $4n^3$
  end for
  Compute the upper bound on the eigenvalues of the Hessian of the cost function: $\Delta = \frac{1}{K} \sum_{j=1}^{K} \frac{\log c_j}{2} \coth(\frac{\log c_j}{2})$; ▷ # $5K$
  Compute stepsize $\alpha_k = \alpha(\Delta)$;
  Compute $\operatorname{grad} f(x_k) = \frac{1}{K} \sum_{i=1}^{K} G_i$; ▷ # $(K+1)n^2$
  Compute $x_{k+1} = R_{x_k}(-\alpha_k \operatorname{grad} f(x_k))$;
  $k = k + 1$;
end while

2.4.2 Riemannian Barzilai-Borwein method with line search

The Barzilai-Borwein (BB) stepsize makes implicit use of second order information of the cost function, see [31, 52] for details. The most two frequently used versions of the BB stepsize are

$$\alpha_{k+1}^{\mathrm{BB1}} = \frac{g(s_k, s_k)}{g(s_k, y_k)}, \tag{2.4.3}$$

$$\alpha_{k+1}^{\mathrm{BB2}} = \frac{g(s_k, y_k)}{g(y_k, y_k)}, \tag{2.4.4}$$

where $s_k = \mathcal{T}_{\alpha_k \eta_k}(\alpha_k \eta_k)$, $y_k = \operatorname{grad} f(x_{k+1}) - \mathcal{T}_{\alpha_k \eta_k}(\operatorname{grad} f(x_k))$, and $g(s_k, y_k) > 0$. An adaptive BB stepsize selection rule given in [39], denoted by $\mathrm{ABB}_{\min}$, is defined as

$$\alpha_{k+1}^{\mathrm{ABB}_{\min}} = \begin{cases} \min\{\alpha_j^{\mathrm{BB2}} : j = \max(1, k - m_a), \ldots, k\}, & \text{if } \dfrac{\alpha_{k+1}^{\mathrm{BB2}}}{\alpha_{k+1}^{\mathrm{BB1}}} < \tau \\ \alpha_{k+1}^{\mathrm{BB1}}, & \text{otherwise} \end{cases} \tag{2.4.5}$$

where m_a is a nonnegative integer and $\tau \in (0,1)$. $\alpha_{k+1}^{\mathrm{ABB}_{\min}}$ tends to compute small BB2 stepsizes, spaced out with some BB1 stepsizes. Notice that when $\tau = 1$, $\alpha_{k+1}^{\mathrm{ABB}_{\min}}$ is the smallest BB2 stepsize in several previous iterations.

In [52], RBB based on (2.4.3) has been applied to the SPD Karcher mean computation. We summarize their implementation in Algorithm 4, which uses an extrinsic representation of a tangent vector, exponential mapping (2.3.4) and parallel translation (2.3.10). Algorithm 5 states our implementation of RBB using the intrinsic representation approach, retraction (2.3.5) and vector transport by parallelization (2.3.12). We present the number of flops for each step on the right-hand side of the algorithms, except problem-related operations, i.e., function, gradient evaluations and line search procedure. Note that Step 7 and Step 11 in Algorithm 4 share the common term $\exp(\alpha_k x_k^{-1}\eta_k)$, which dominates the computational time and is only computed once. Having $w = n^2$ and $d = n(n+1)/2$, the number of flops per iteration for Algorithm 4 and Algorithm 5 are $103n^3/3 + o(n^3)$ and $22n^3/3 + o(n^3)$ respectively. The number of flops required by Algorithm 5 is smaller than that of Algorithm 4, and the computational efficiency mainly comes from the choice of retraction and vector transport, and the fact that Riemannian metric reduces to the Euclidean metric using the intrinsic representation of a tangent vector.

2.4.3 A Limited-memory Riemannian BFGS

In this section, we employed a limited-memory RBFGS method (LRBFGS) for the SPD Karcher mean computation. The limited-memory BFGS method is based on the BFGS method which stores and transports the inverse Hessian approximation as a dense matrix. Specifically, the search direction in RBFGS is $\eta_k = -\mathcal{B}_k^{-1} \operatorname{grad} f(x_k)$, where $\mathcal{B}_k$ is a linear operator that approximates the action of the Hessian on $\mathrm{T}_{x_k}\mathcal{M}$, and requires a rank-two update at each iteration, see [51, Algorithm 1] for the update formula. Unlike BFGS, its limited-memory version stores only some relatively small number of vectors that represent the approximation implicitly. Therefore LRBFGS is appropriate for large-size problems, due to its benefit in reducing storage requirements and computation time per iteration.

Our implementation of LRBFGS depends on [51, Algorithm 2]. We modified this version to use an alternate update from [49] that allows the line search using the Riemannian Wolfe conditions to be replaced by the Armijo line search used in [2, Section 4.2]. We provide LRBFGS for the SPD Karcher mean computation in Algorithm 6 and 7. In fact, those two algorithms are ready

Algorithm 4 RBB for the SPD Karcher mean computation using extrinsic representation [52]

Input: backtracking reduction factor $\varrho \in (0,1)$; Armijo parameter $\delta \in (0,1)$; initial iterate $x_0 \in \mathcal{S}^{\mathrm{n}}_{++}$; the first stepsize α_0^{BB};

1: $k = 0$;
2: Compute $f(x_k)$, $\operatorname{grad} f(x_k)$;
3: **while** $\|\operatorname{grad} f(x_k)\| > \epsilon$ **do**
4: Set stepsize $\alpha_k = \alpha_k^{BB}$;
5: Set $\eta_k = -\operatorname{grad} f(x_k)$; ▷ # w
6: If $\|\operatorname{grad}\operatorname{grad} f(x_k)\|/\|\operatorname{grad} f(x_0)\| < accuracy$
then set $x_{k+1} = x_k \exp(\alpha_k x_k^{-1}\eta_k)$ and go to Step 10;
7: Compute $\tilde{x}_k = x_k \exp(\alpha_k x_k^{-1}\eta_k)$; ▷ # $61n^3/3$
8: If $f(\tilde{x}_k) \leq f(x_k) + \delta\alpha_k g(\operatorname{grad} f(x_k), \eta_k)$,
then set $x_{k+1} = \tilde{x}_k$ and go to Step 10;
9: Set $\alpha_k = \varrho\alpha_k$ and go to Step 7;
10: Compute $\operatorname{grad} f(x_{k+1})$;
11: Compute $s_k = \alpha_k\eta_k \exp(\alpha_k x_k^{-1}\eta_k)$, $y_k = \operatorname{grad} f(x_{k+1}) + \eta_k \exp(\alpha_k x_k^{-1}\eta_k)$; ▷ # $2n^3 + n^2$
12: Compute $\alpha_{k+1}^{BB} = g(s_k, s_k)/g(s_k, y_k)$; ▷ # $12n^3$
13: Set $\alpha_{k+1}^{BB} = \min\{\alpha_{max}, \max\{\epsilon, \alpha_{k+1}^{BB}\}\}$ if $g(s_k, y_k) > 0$; otherwise, $\alpha_{k+1}^{BB} = \alpha_{max}$;
14: $k = k + 1$;
15: **end while**

to solve any optimization problems on $\mathcal{S}^{\mathrm{n}}_{++}$ as long as the readers provide a cost function and its Riemannian gradient. Algorithm 6 uses the extrinsic representation and general vector transport. Algorithm 7 uses the intrinsic representation and vector transport by parallelization. The number of flops for each step is given on the right-hand side of the algorithms. For simplicity of notation, we use λ_m, λ_r, and λ_t to denote the flops in the metric, retraction, and vector transport evaluation respectively, and use superscripts, w and d, to denote the extrinsic and intrinsic representations respectively. The numbers of flops per iteration for Algorithm 6 and Algorithm 7, respectively, are

$$\#^w = 2(l+2)\lambda_m^w + 4lw + \lambda_r^w + 4w + 2(l+1)\lambda_t^w \tag{2.4.6}$$

$$\#^d = 2(l+2)\lambda_m^d + 4ld + \lambda_r^d + 4d + (13n^3/3 + 2d) \tag{2.4.7}$$

Notice that m is the upper limit of the limited-memory size l. Also notice that there is no λ_t^d term in equation (2.4.7) since the vector transport by parallelization (2.3.12) is used, which is the identity. The last term $(13n^3/3 + 2d)$ in (2.4.7) comes from the evaluation of functions $E2D$ and

Algorithm 5 RBB for the SPD Karcher mean computation using intrinsic representation and vector transport by parallelization

Input: backtracking reduction factor $\varrho \in (0,1)$; Armijo parameter $\delta \in (0,1)$; initial iterate $x_0 \in \mathcal{M}$; the first stepsize α_0^{BB};

1: $k = 0$;
2: Compute $\operatorname{grad} f(x_k)$;
3: Compute $x_k = L_k L_k^T$; ▷ $\#n^3/3$
4: Compute $\mathrm{gf}_k^d = E2D_{x_k}(\operatorname{grad} f(x_k))$ by Algorithm 1; ▷ $\#\ 2n^3 + d$
5: **while** $\|\mathrm{gf}_k^d\| > \epsilon$ **do**
6: Set stepsize $\alpha_k = \alpha_k^{BB}$;
7: Set $\eta_k = -\mathrm{gf}_k^d$; ▷ $\#d$
8: Compute $\eta_k^w = D2E_{x_k}(\eta_k)$ by Algorithm 2; ▷ $\#2n^3 + n(n+1)/2$
9: If $\|\mathrm{gf}_k^d\|/\|\mathrm{gf}_0^d\| < accuracy$
then set $x_{k+1} = R_{x_k}(\alpha_k \eta_k^w)$ using (2.3.5) and go to Step 13;
10: Compute $\tilde{x}_k = R_{x_k}(\alpha_k \eta_k^w)$ using (2.3.5) ; ▷ $\#\ 3n^3 + 3n^2$
11: If $f(\tilde{x}_k) \leq f(x_k) + \delta \alpha_k \eta_k^T \mathrm{gf}_k^d$,
then set $x_{k+1} = \tilde{x}_k$ and go to Step 13;
12: Set $\alpha_k = \varrho \alpha_k$ and go to Step 10;
13: Compute $\operatorname{grad} f(x_{k+1})$;
14: Compute $x_{k+1} = L_{k+1} L_{k+1}^T$; ▷ $\#n^3/3$
15: Compute $\mathrm{gf}_k^d = E2D_{x_k}(\operatorname{grad} f(x_k))$ by Algorithm 1; ▷ $\#\ 2n^3 + d$
16: Compute $s_k = \alpha_k \eta_k$, $y_k = \mathrm{gf}_{k+1}^d - \mathrm{gf}_k^d$; ▷ $\#2d$
17: Compute $\alpha_{k+1}^{BB} = s_k^T s_k / s_k^T y_k$; ▷ $\#\ 4d$
18: Set $\alpha_{k+1}^{BB} = \min\{\alpha_{max}, \max\{\epsilon, \alpha_{k+1}^{BB}\}\}$ if $g(s_k, y_k) > 0$; otherwise, $\alpha_{k+1}^{BB} = \alpha_{max}$;
19: $k = k + 1$;
20: **end while**

$D2E$ given in Algorithm 1 and 2. For the metric evaluation using different representations, we have $\lambda_m^w = 6n^3 + o(n^3)$ and $\lambda_m^d = n^2 + o(n^2)$. Simplifying and rearranging (2.4.6) and (2.4.7), we have

$$\#^w = 12ln^3 + 24n^3 + \lambda_r^w + 2(l+1)\lambda_t^w + o(ln^3) + o(n^3) \tag{2.4.8}$$

$$\#^d = 4ln^2 + 13n^3/3 + \lambda_r^d + o(ln^2) + o(n^3). \tag{2.4.9}$$

From (2.4.8) and (2.4.9), the computational benefit of the intrinsic representation is substantial. The limited-memory size l imposes a much heavier burden on Algorithm 6 where the extrinsic representation is used. In our implementation of Algorithm 7, we suggest retraction (2.3.5), which

needs $3n^3 + o(n^3)$ flops. Hence the overall flops required by Algorithm 7 is $\#^d = 4ln^2 + 22n^3/3 + o(ln^2) + o(n^3)$. Notice that if the locking condition is imposed on Algorithm 7, extra $12(l+1)n^2$ flops are needed. For Algorithm 6, any choice of retraction and vector transport would yield a larger flop compared to Algorithm 7.

However, our complexity analysis above focuses on manifold- and algorithm-related operations, the problem-related operations—function, gradient evaluations and line search—are not considered. From the discussion in (2.3.3), the evaluation of cost function requires $18Kn^3$ flops, and the computation of gradient requires extra $5Kn^3$ flops given the function evaluation. The line search procedure may take a few steps to terminate, and each step requires one cost function evaluation. In the ideal case where the initial stepsize satisfies the Armijo condition in Step 20 in Algorithm 7, i.e., the cost function is evaluated only once, the flops required by problem-related operations is $23Kn^3$. As n gets larger, the proportion of computational time spent on function and gradient evaluations is

$$\frac{23Kn^3}{23Kn^3 + 4ln^2 + 22n^3/3} \approx \frac{23K}{23K + 22/3} \tag{2.4.10}$$

$$\geq \frac{23 \cdot 3}{23 \cdot 3 + 22/3} \approx 90.39\%. \tag{2.4.11}$$

Inequality (2.4.11) comes from the fact that $K \geq 3$ and (2.4.10) is an increasing function of K. It is shown that the problem-related operations dominate the computation time for high dimensional matrices, which is consistent with our empirical observations in experiments that $80\% - 90\%$ timing comes from function and gradient evaluations. If the line search procedure requires more steps to terminate, then the problem-related operations would result in a larger proportion of total computational time. Therefore, it is crucial to have a good initial stepsize.

Finally, note that LRBFGS with zero memory size, i.e., $m = 0$, is equivalent to RBB. This is easy to verify by setting $m = 0$ in Algorithm 6 and 7. Just like there are different versions of the BB stepsize, different alternatives are available for the initial scaling γ_{k+1} in step 22 of Algorithm 6 and step 27 of Algorithm 7, such as BB1 (2.4.3), BB2 (2.4.4), and $\text{ABB}_{\min}$ (2.4.5). In particular, we use BB2 (2.4.4) as the default.

[6] If the locking condition is imposed, then $y_k^{(k+1)} = \operatorname{grad} f(x_{k+1})/\beta_k - \mathcal{T}_{\alpha_k \eta_k} \operatorname{grad} f(x_k)$, where $\beta_k = \|\alpha_k \eta_k\| / \|\mathcal{T}_{R_{\alpha_k \eta_k}} \alpha_k \eta_k\|$.

[7] If retraction (2.3.5) and isometric vector transport (2.3.15) that satisfy the locking condition are used, s_k and y_k are computed as follows: compute $z^w = \mathcal{T}_{R_{\alpha_k \eta_k^w}}(\alpha_k \eta_k^w)$, where $\mathcal{T}_{R_\xi} \eta = \eta + (\eta X^{-1} \xi + \xi X^{-1} \eta)/2$; obtain the

2.5 Numerical experiments

In this section, we compare the performance of LRBFGS described in Algorithm 7 and existing state-of-the-art methods, including the Riemannian Barzilai-Borwein method (RBB) provided in [52] (using implementation in Algorithm 5), the Riemannian steepest descent method with stepsize selection rule proposed by Q. Rentmeesters et al. (RSD-QR) in [82, Section 3.6], the Richardson-like iteration (RL) of [16], and the Riemannian BFGS method (RBFGS) presented in [49, 51].

All experiments are performed on the Florida State University HPC system using Quad-Core AMD Opteron(tm) Processor 2356 2.3GHz. Experiments in Section (2.5.2) are carried out using C++, compiled with gcc-4.7.x. Section 2.5.4 presents a comparison of computation time between MATLAB and C++ implementations. All MATLAB experiments are performed using MATLAB R2015b (8.6.0.267246) 64-bit (glnxa64). In particular, we use the MATLAB implementation of RL in Bini et al.'s Matrix Means Toolbox[1].

Regarding the parameter setting, we set Armijo parameter $\delta = 10^{-4}$, backtracking reduction factor $\varrho = 0.5$ for well-conditioned data sets and $\varrho = 0.25$ for ill-conditioned ones, maximum stepsize $\alpha_{max} = 100$, and minimum stepsize α_{min} is machine epsilon. We mention here that it is found from our experiments (not shown in this paper) that LRBFGS is much less sensitive to ϱ that RBB. We use $\varrho = 0.25$ since it leads to the best performance of RBB, while LRBFGS behaves similarly for different values of ϱ. The initial stepsize in the first iteration is chosen by the strategy in [82], i.e., $\alpha_0 = 2/(1+U)$, where U is the upper bound at the initial iterate defined in inequality (2.2.3). For LRBFGS, we use different memory sizes m as specified in the legends of the figures, and impose the locking condition for ill-conditioned matrices. Specifically, we impose the locking condition on LRBFGS in the bottom plots of Figure 2.1, 2.2, 2.3, and 2.4. As we have shown in Section 2.4, imposing the locking condition requires extra complexity. For well-conditioned data sets, the problem is easy to handle, and the locking condition is not necessary. While in the ill-conditioned case, imposing the locking condition can reduce the number of iterations. The extra time caused by the locking condition is smaller than the time saved by a reduction in the number of function and gradient evaluations. The benefit of the locking condition is also demonstrated

intrinsic representation z of z^w by Algorithm 1; compute $\beta = \alpha_k^2 \eta_k^T \eta_k / z^T z$, $v_1 = 2\alpha_k \eta_k$, $v_2 = -\alpha_k \eta_k - \beta z$; Define $s_k = (I - 2v_2 v_2^T / v_2^T v_2)(I - 2v_1 v_1^T / v_1^T v_1)(\alpha_k \eta_k)$, $y_k = \mathrm{gf}_{k+1}^d / \beta - (I - 2v_2 v_2^T / v_2^T v_2)(I - 2v_1 v_1^T / v_1^T v_1)\mathrm{gf}_k^d$.

in [49]. In order to achieve sufficient accuracy, we skip the line search procedure when the iterate is close enough to the minimizer by setting $accuracy = 10^{-5}$ in Algorithm 5 and 7. Unless otherwise specified, our choice of the initial iterate is the Arithmetic-Harmonic mean of data points [54]. We run the algorithms until they reach their highest accuracy.

For simplicity of notation, throughout this section we denote the number, dimension, and condition number of the matrices by K, n, and κ respectively. For each choice of (K, n) and the range of conditioning desired, a single experiment comprises generating 5 different sets of K random $n \times n$ matrices with appropriate condition numbers, and running all 5 algorithms on each set with identical parameters. The result of the experiment is the distance to the true Karcher mean averaged over the 5 sets as a function of iteration and time. To obtain sufficiently stable timing results, an average time is taken of several runs for a total runtime of at least 1 minute.

2.5.1 Experiment design

When defining each set of experiments, we choose a desired (true) Karcher mean μ, and construct data matrices A_i's such that their Karcher mean is exactly μ, i.e., $\sum_{i=1}^{K} \mathrm{Exp}_{\mu}^{-1}(A_i) = 0$ holds. The benefits of this scheme are: (i) We can control the conditioning of μ and A_i's, and observe the influence of the conditioning on the performance of algorithms. (ii) Since the true Karcher mean μ is known, we can monitor the distance δ between μ and the iterates produced by various algorithms, thereby removing the need to consider the effects of termination criteria.

Given a Karcher mean μ, the A_i's are constructed as follows: (i) Generate W_i in Matlab, with n the size of matrix, f the order of magnitude of the condition number, and p some number less than n,

```
[O, ~] = qr(randn(n));
D = diag([rand(1,p)+1, (rand(1,n-p)+1)*10^(-f)]);
W = O * D * O'; W = W/norm(W,2);
```

(ii) Compute $\eta_i = \mathrm{Exp}_{\mu}^{-1}(W_i)$. (iii) Enforce the condition $\sum_{i=1}^{K} \eta_i = 0$ on η_i's. Specifically, we test on data sets with $K = 3,\ 30,\ 100$. In the case of $K = 3$, we enforce $\eta_3 = -\eta_1 - \eta_2$. When $K = 30$ or $K = 100$, let $k_i = 5(k-1) + i$ for $1 \le k \le K/5$ and $1 \le i \le 5$. We enforce $\eta_{k_4} = -\eta_{k_1} - 0.5\eta_{k_3}$ and $\eta_{k_5} = -\eta_{k_2} - 0.5\eta_{k_3}$, which gives $\sum_{i=1}^{5} \eta_{k_i} = 0$, and thus $\sum_{k=1}^{K/5} \sum_{i=1}^{5} \eta_{k_i} = 0$. (iv) Compute $A_i = \mathrm{Exp}_{\mu}(\eta_i)$.

Note that instead of producing η_i directly, we produce W_i first and obtain η_i from the log-mapping, since this procedure gives us greater control over the conditioning of data points. As discussed in Section 2.2, we can take the Karcher mean μ as the identity matrix in numerical experiments, so we "translate" the data set $\{\mu, A_1, \ldots, A_K\}$ to $\{I, L^{-1}A_1L^{-T}, \ldots, L^{-1}A_KL^{-T}\}$ using an isometry, where $\mu = LL^T$ as the final step.

2.5.2 Comparison of performances between different algorithms using C++

We now compare the performances of all 5 algorithms on various data sets by examining results from representative experiments for different choices of (K, n, κ).

Figure 2.1 displays the performance results of different algorithms running on small-size problems, taking $K = 3$ and $n = 3$. Both well-conditioned ($1 \leq \kappa(A_i) \leq 20$) and ill-conditioned ($10^5 \leq \kappa(A_i) \leq 10^{10}$) data sets are tested. In the well-conditioned case, it is seen that all 5 algorithms are comparable in terms of computation time even though they require different numbers of iterations. For ill-conditioned matrices, RL and RSD-QR require significantly more iterations, but they are still efficient in terms of timing due to the low computational cost per iteration. RBB and LRBFGS require similar numbers of iterations, but LRBFGS with $m > 0$ takes more time. The computational complexity per iteration for Algorithm 7 with the locking condition is $23Kn^3 + 22n^3/3 + 16ln^2 + o(ln^2) + o(n^3)$ as discussed in Section 2.4. So when the size of the problem is small, the impact of memory size l is visible. But this is not the case when the size of the problem gets larger, as shown in Figure 2.2 and 2.3.

Figure 2.2 and Figure 2.3 report the results of tests conducted on data sets with large K ($K = 100,\ n = 3$) and large n ($K = 30,\ n = 100$) respectively. Note that when $n = 100$, the dimension of $\mathcal{S}^n_{++}$ is $d = n(n+1)/2 = 5050$. In each case, both well- and ill-conditioned data sets are tested. For well-conditioned matrices, we observe that LRBFGS and RBB perform similarly, with a slight advantage for LRBFGS. The advantage of LRBFGS becomes larger as the matrices become increasingly ill-conditioned. Note that RBFGS is very inefficient for large n as expected and shown in Figure 2.3.

As the last test in this section, we compare the performances of the algorithms using two different initial iterates: the Arithmetic-Harmonic mean and the Cheap mean [17]. The Cheap mean is known to be a good approximation of the Karcher mean, but, is not cheap to compute. We use Bini et al.'s Matrix Means Toolbox[1] for the computation of Cheap mean. We consider 30

badly conditioned 30×30 matrices ($10^6 \leq \kappa(A_i) \leq 10^9$). The results are presented in Figure 2.4. Notice that the time required to compute the initial iterate is included in the plots. The x-axis of the bottom-right plot does not start from 0, which shows that the computation of the Cheap mean is time demanding. We observe that the choice of initial iterate is crucial to RBFGS, and it affects the other algorithms in the early steps. When the initial iterate is close enough to the true solution, as shown in the bottom right plot in Figure 2.4, we observe a faster convergence in the first a few steps for all algorithms. In both cases, LRBFGS outperforms the other algorithms in terms of computation time and number of iterations per unit of accuracy required.

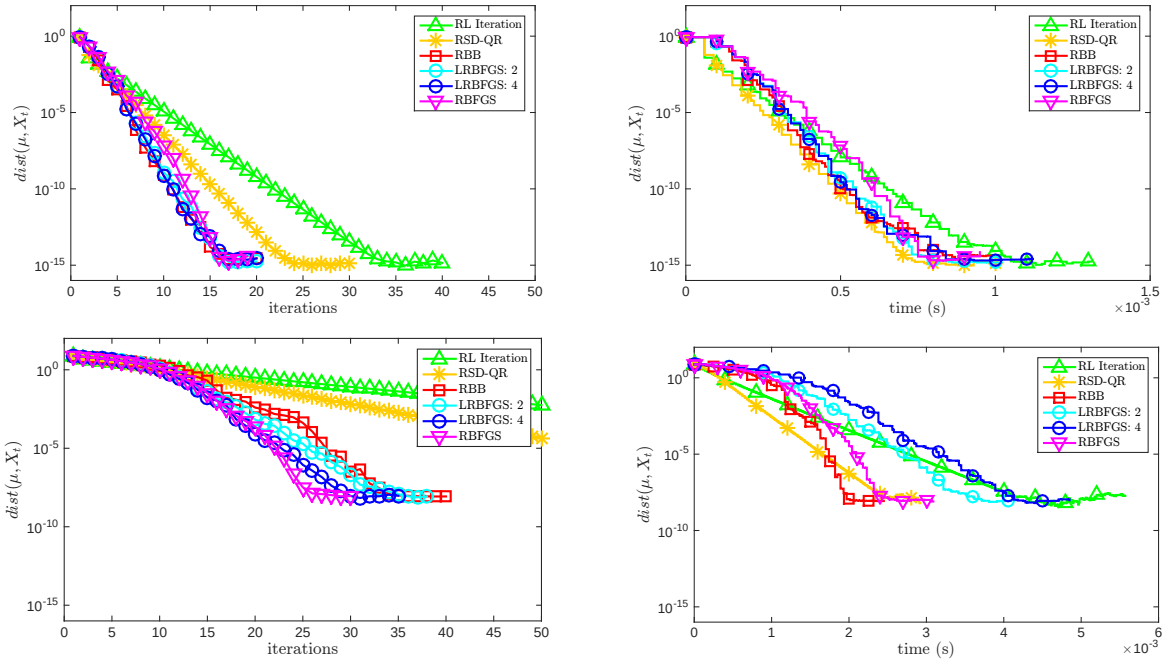


Figure 2.1: Evolution of averaged distance between current iterate and the exact Karcher mean with respect to time and iterations with $K = 3$, $n = 3$. Top: $1 \leq \kappa(A_i) \leq 20$; Bottom: $10^5 \leq \kappa(A_i) \leq 10^{10}$.

2.5.3 Comparison of the Riemannian metric and Euclidean metric

In this section we compare two different metrics, the Euclidean metric and Riemannian metric, for RSD, LRBFGS and RBFGS. RSD refers to the standard steepest descent in [53], and the initial stepsize is taken as the classical strategy in [93, (3.44)]. Notice that LRBFGS with $m = 0$ is RBB, i.e., RSD with the BB stepsize. In this test, we generate the true Karcher mean (minimizer) of

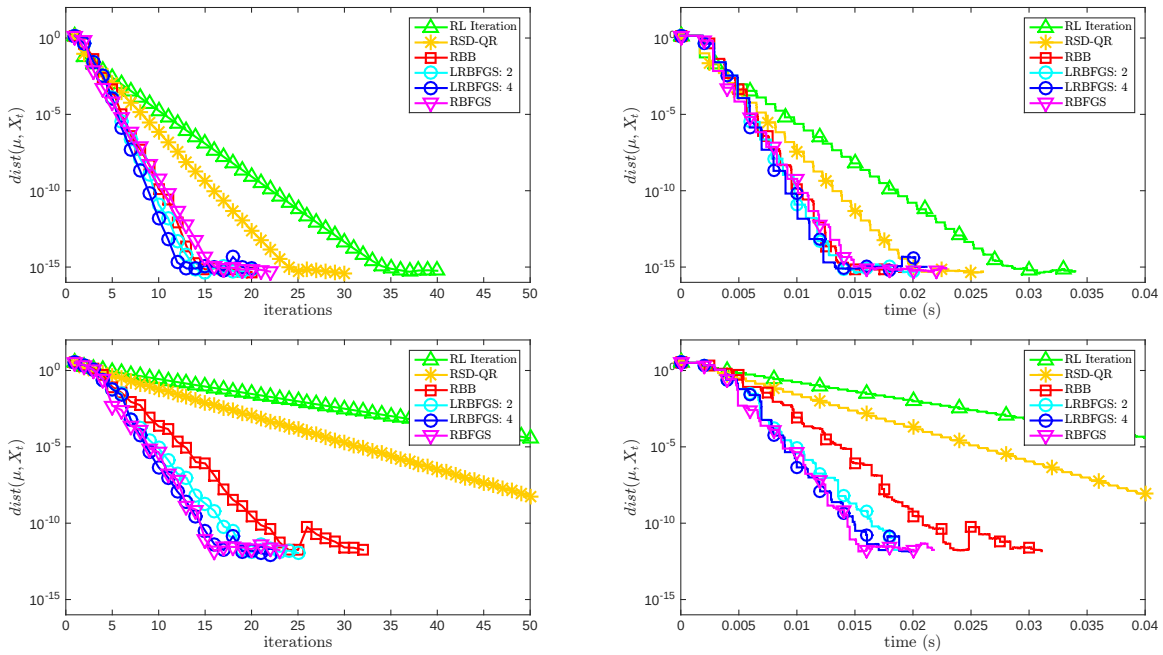


Figure 2.2: Evolution of averaged distance between current iterate and the exact Karcher mean with respect to time and iterations with $K = 100$ and $n = 3$; Top: $1 \leq \kappa(A_i) \leq 200$; Bottom: $10^3 \leq \kappa(A_i) \leq 10^7$.

various sizes as described in Section 2.5.1 with condition number between 10 and 20. The majority ($> 60\%$) of the data matrices in each data set are well-conditioned ($\kappa < 100$). The results are reported in Figure 2.5. In the legends, 'Euc' refers to the Euclidean metric and 'Rie' refers to the Riemannian metric. We observe that the Riemannian metric shows more than two hundreds times faster convergence speed for RSD. When the Riemannian metric is used, LRBFGS with $m = 0$ and $m = 2$ behave similarly just as the well-conditioned case in Section 2.5.2. However, in the case of the Euclidean metric, LRBFGS with $m = 2$ is much faster than $m = 0$. For RBFGS, the influence of the metric becomes less significant compared to simpler methods.

2.5.4 Comparison between C++ and MATLAB implementations

In this section, we compare the computation time of the algorithms on the SPD Karcher mean computation implemented by C++ and MATLAB. The results are reported in Figure 2.6. The first column indicates that the C++ and MATLAB implementations are identical in terms of iterations. The second column displays the log-log plots of computation time vs. average distance between

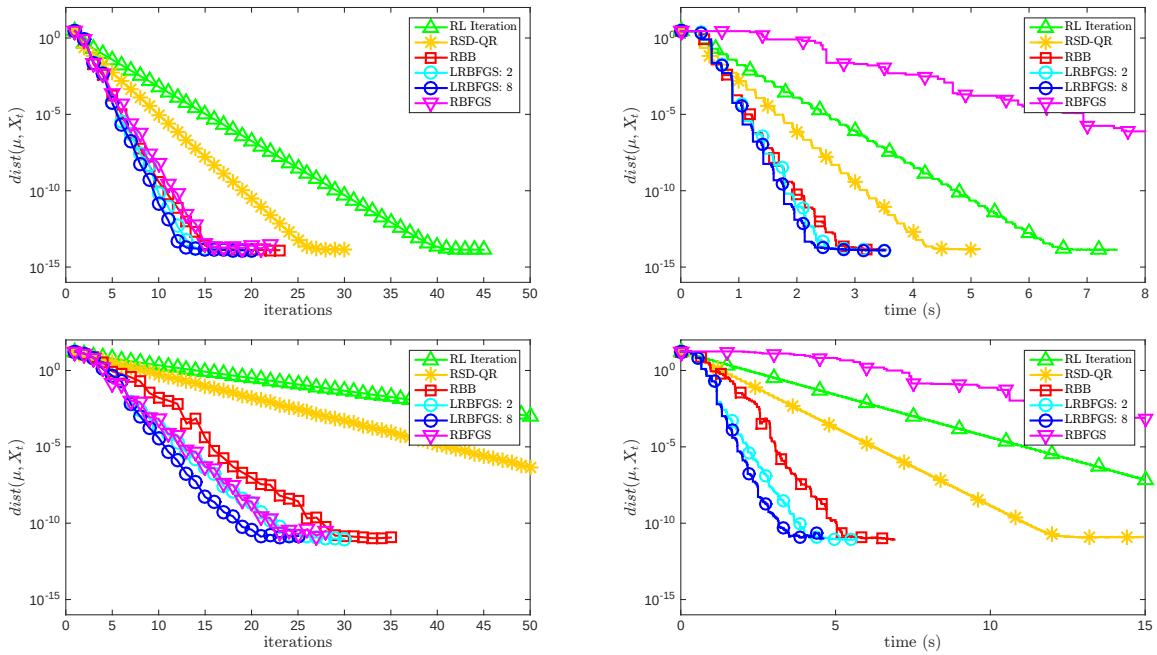


Figure 2.3: Evolution of averaged distance between current iterate and the exact Karcher mean with respect to time and iterations with $K = 30$ and $n = 100$; Top: $1 \leq \kappa(A_i) \leq 20$; Bottom: $10^4 \leq \kappa(A_i) \leq 10^7$.

each iterate and the exact Karcher mean. For small-size problem, the C++ implementation is faster than that of MATLAB by a factor of 100 or more with the factor gradually reducing as n or K gets larger. This phenomenon can be explained by the fact that when n or K is small, the difference of efficiency between C++ and Matlab implementations is mainly due to the difference between compiled languages and interpreted languages. When n or K gets larger, the BLAS and LAPACK calls start to dominate the computation time, which leads to a decrease in the factor. Note that we implemented LRBFGS and RBB as a user-friendly library. It is observed that the overhead of MATLAB library machinery dominates the computation time for $k = 3$ and $n = 3$, but it becomes negligible for large-size problems.

2.6 Conclusions

In this chapter, we present an LRBFGS method for the SPD Karcher mean computation, and provide efficient implementation techniques. There are several alternatives from which to choose

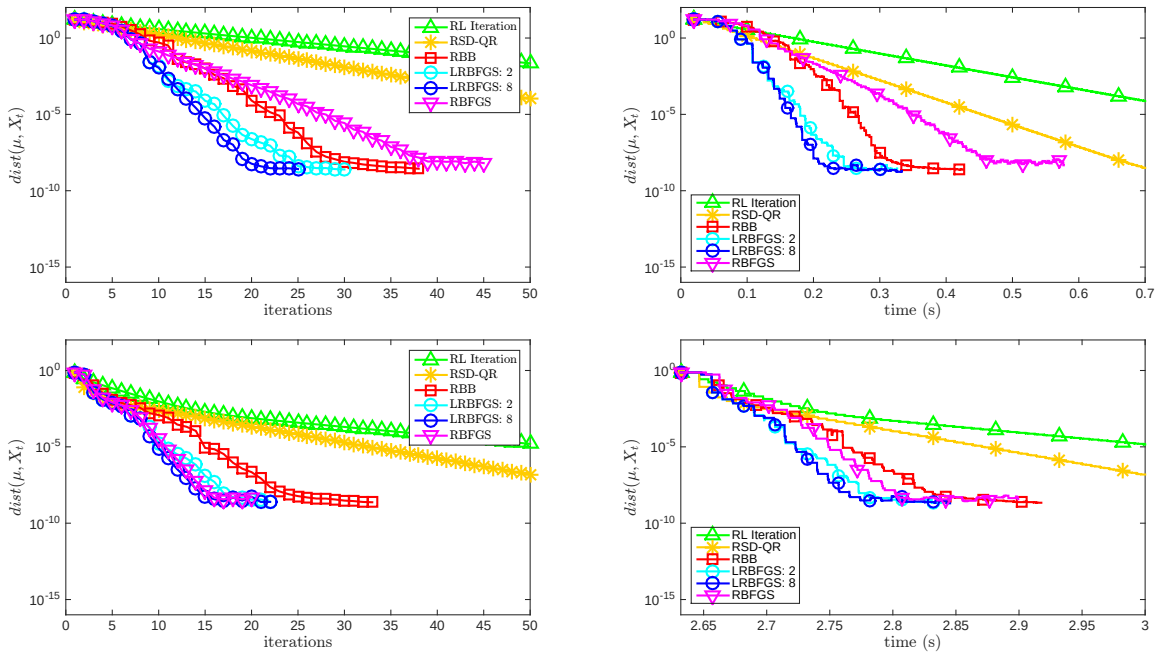


Figure 2.4: Comparison of different algorithms using different initial iterates with $K = 30$, $n = 30$, and $10^6 \leq \kappa(A_i) \leq 10^9$. Top: using the Arithmetic-Harmonic mean as initial iterate; Bottom: using the Cheap mean as initial iterate.

for the representation of a tangent vector, retraction and vector transport. We provide complexity-based recommendations for those alternatives.

Our numerical experiments provide empirical guidelines to choose between various methods and two metrics. It is observed that RSD-QR and RL perform very well when n and K are small, and RSD-QR systematically outperforms RL. As n or K gets larger, RSD-QR and RL become less appealing, while RBB and LRBFGS single out. We recommend using LRBFGS as the default method for the SPD Karcher mean computation mainly for three reasons: (i) When the data matrices are well-conditioned, LRBFGS and RBB are competitive, with a slight advantage for LRBFGS on some test sets. (ii) As the data matrices get ill-conditioned, LRBFGS outperforms RBB. (iii) The performance of RBB depends on the choice of parameters, such as the reduction factor ϱ in the back tracking line search procedure, while LRBFGS is much less sensitive to parameter choices. Since RBB is in fact LRBFGS with $m = 0$, that is to say, LRBFGS can benefit from choosing $m > 0$.

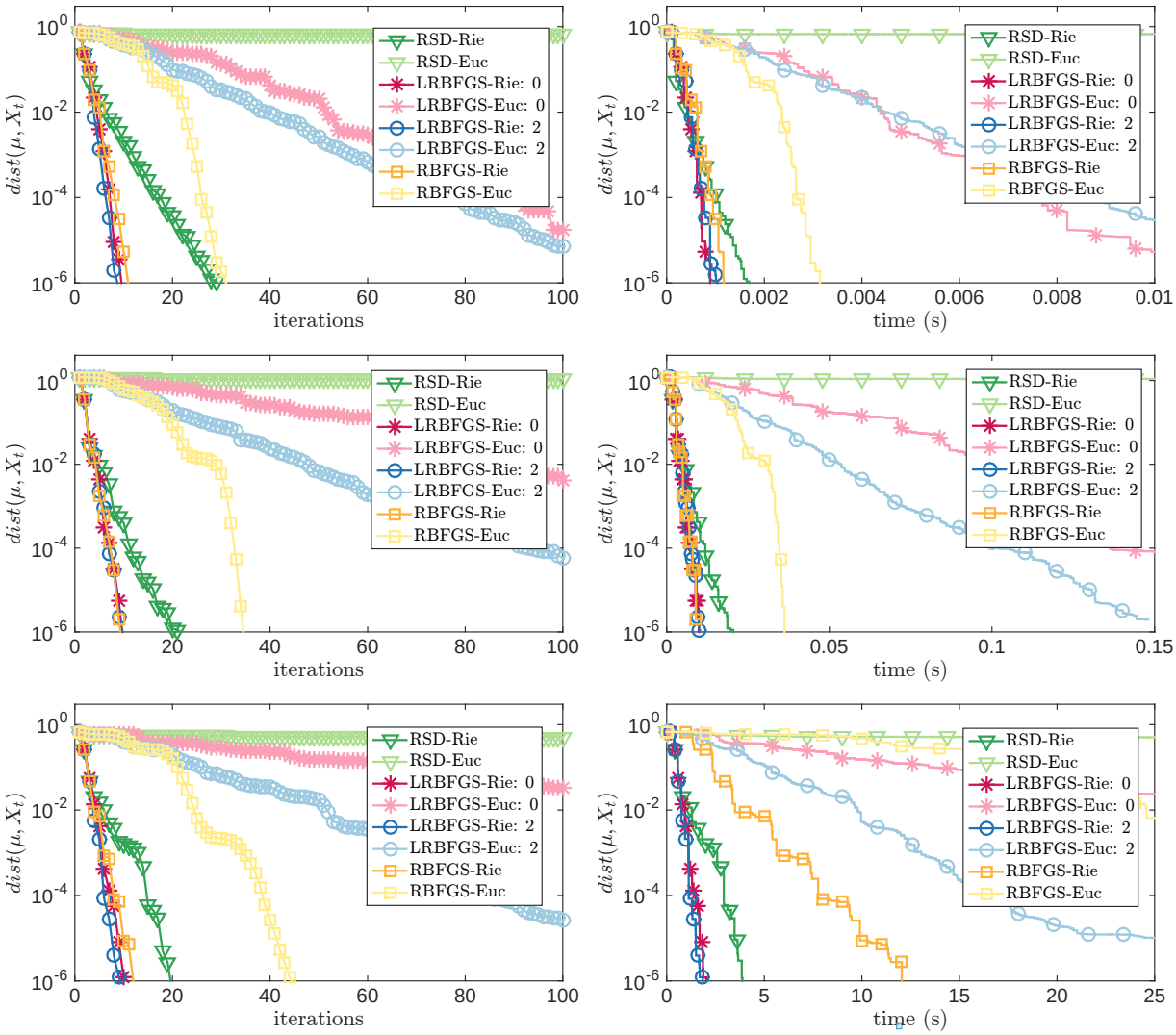


Figure 2.5: Comparison of different algorithms using Riemannian metric and Euclidean metric. Top row: $K = 3$, $n = 3$, and $1 \leq \kappa(A_i) \leq 10^4$; Middle row: $K = 100$, $n = 3$, and $1 \leq \kappa(A_i) \leq 10^6$; Bottom: $K = 30$, $n = 100$, and $1 \leq \kappa(A_i) \leq 10^5$.

We also present empirical illustration of the speedup of C++ implementation compared with MATLAB implementation. Notice that it is demonstrated theoretically and empirically that for large-size problems, the dominant computation time (70% - 90%) is in the problem-related operations, i.e., function and gradient evaluations, and our implementations of manifold- and algorithm-related objects is the cheapest we can get so far.

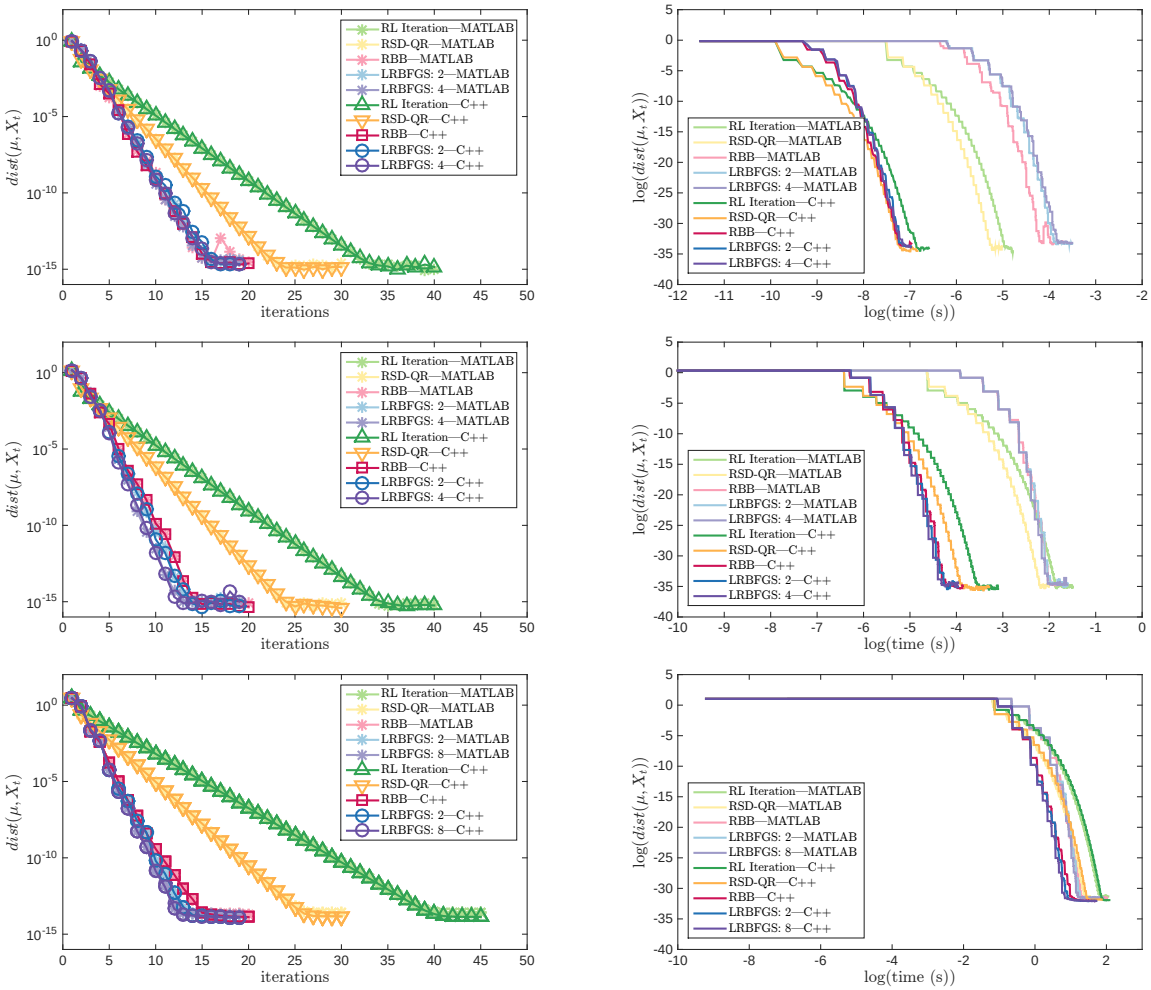


Figure 2.6: Comparison between C++ and MATLAB implementations with different choices of (K, n, κ). Top row: $K = 3$, $n = 3$, and $1 \leq \kappa(A_i) \leq 20$; Middle row: $K = 100$, $n = 3$, and $1 \leq \kappa(A_i) \leq 20$; Bottom: $K = 30$, $n = 100$, and $1 \leq \kappa(A_i) \leq 20$.

Algorithm 6 LRBFGS for problems on $\mathcal{S}_{++}^{\mathrm{n}}$ using extrinsic representation

Input: backtracking reduction factor $\varrho \in (0,1)$; Armijo parameter $\delta \in (0,1)$; initial iterate $x_0 \in \mathcal{M}$; an integer $m > 0$;

1: $k = 0$, $\gamma_0 = 1$, $l = 0$, compute $\operatorname{grad} f(x_k)$;
2: **while** $\|\operatorname{grad} f(x_k)\| > \epsilon$ **do**
3: $\mathcal{H}_k^0 = \gamma_k \operatorname{id}$. Obtain $\eta_k \in \mathrm{T}_{x_k}\mathcal{M}$ by the following algorithm, Step 4 to Step 13:
4: $q \leftarrow \operatorname{grad} f(x_k)$;
5: **for** $i = k-1, k-2, \ldots, k-l$ **do** ▷ # $l(\lambda_m^w + 2w)$
6: $\xi_i \leftarrow \rho_i g(s_i^{(k)}, q)$; $q \leftarrow q - \xi_i y_i^{(k)}$;
7: **end for**
8: $r \leftarrow \mathcal{H}_k^0 q$; ▷ # w
9: **for** $i = k-l, k-l+1, \ldots, k-1$ **do** ▷ # $l(\lambda_m^w + 2w)$
10: $\omega \leftarrow \rho_i g(y_i^{(k)}, r)$;
11: $r \leftarrow r + s_i^{(k)}(\xi_i - \omega)$;
12: **end for**
13: Set $\eta_k = -r$, $\alpha_k = 1$; ▷ # w
14: If $$\|\operatorname{grad} f(x_k)\| / \|\operatorname{grad} f(x_0)\| < accuracy$$ then set $x_{k+1} = R_{x_k}(\alpha_k \eta_k)$ and go to Step 18; ▷ # λ_r^w
15: Compute $\tilde{x}_k = R_{x_k}(\alpha_k \eta_k)$; ▷ # λ_r^w
16: If $$f(\tilde{x}_k) \le f(x_k) + \delta \alpha_k g(\operatorname{grad} f(x_k), \eta_k),$$ then set $x_{k+1} = \tilde{x}_k$ and go to Step 18;
17: Set $\alpha_k = \varrho \alpha_k$ and go to Step 15;
18: Compute $\operatorname{grad} f(x_{k+1})$;
19: Define $s_k^{(k+1)} = \mathcal{T}_{\alpha_k \eta_k} \alpha_k \eta_k$ and $y_k^{(k+1)} = \operatorname{grad} f(x_{k+1}) - \mathcal{T}_{\alpha_k \eta_k} \operatorname{grad} f(x_k)$; [6] ▷ # $2\lambda_t^w + 2w$
20: Compute $a = g(y_k^{(k+1)}, s_k^{(k+1)})$ and $b = \|s_k^{(k+1)}\|^2$; ▷ # $2\lambda_m^w$
21: **if** $\frac{a}{b} \ge 10^{-4} \|\operatorname{grad} f(x_k)\|$ **then** ▷ # λ_m^w
22: Compute $c = \|y_k^{(k+1)}\|^2$ and define $\rho_k = 1/a$ and $\gamma_{k+1} = a/c$; ▷ # λ_m^w
23: Add $s_k^{(k+1)}$, $y_k^{(k+1)}$ and ρ_k into storage and if $l \ge m$, then discard vector pair $\{s_{k-l}^{(k)}, y_{k-l}^{(k)}\}$ and scalar ρ_{k-l} from storage, else $l \leftarrow l+1$; Transport $s_{k-l+1}^{(k)}, s_{k-l+2}^{(k)}, \ldots, s_{k-1}^{(k)}$ and $y_{k-l+1}^{(k)}, y_{k-l+2}^{(k)}, \ldots, y_{k-1}^{(k)}$ from $\mathrm{T}_{x_k}\mathcal{M}$ to $\mathrm{T}_{x_{k+1}}\mathcal{M}$ by $\mathcal{T}$, then get $s_{k-l+1}^{(k+1)}, s_{k-l+2}^{(k+1)}, \ldots, s_{k-1}^{(k+1)}$ and $y_{k-l+1}^{(k+1)}, y_{k-l+2}^{(k+1)}, \ldots, y_{k-1}^{(k+1)}$; ▷ # $2(l-1)\lambda_t^w$
24: **else**
25: Set $\gamma_{k+1} \leftarrow \gamma_k$, $\{\rho_k, \ldots, \rho_{k-l+1}\} \leftarrow \{\rho_{k-1}, \ldots, \rho_{k-l}\}$, $\{s_k^{(k+1)}, \ldots, s_{k-l+1}^{(k+1)}\} \leftarrow \{\mathcal{T}_{\alpha_k \eta_k} s_{k-1}^{(k)}, \ldots, \mathcal{T}_{\alpha_k \eta_k} s_{k-l}^{(k)}\}$ and $\{y_k^{(k+1)}, \ldots, y_{k-l+1}^{(k+1)}\} \leftarrow \{\mathcal{T}_{\alpha_k \eta_k} y_{k-1}^{(k)}, \ldots, \mathcal{T}_{\alpha_k \eta_k} y_{k-l}^{(k)}\}$; ▷ # $2l\lambda_t^w$
26: **end if**
27: $k = k+1$;
28: **end while**

Algorithm 7 LRBFGS for problems on $\mathcal{S}^{\mathrm{n}}_{++}$ using intrinsic representation

Input: backtracking reduction factor $\varrho \in (0,1)$; Armijo parameter $\delta \in (0,1)$; initial iterate $x_0 \in \mathcal{M}$; an integer $m > 0$;

1: $k = 0$, $\gamma_0 = 1$, $l = 0$;
2: Compute $\operatorname{grad} f(x_k)$;
3: Compute $\mathrm{gf}_k^d = E2D_{x_k}(\operatorname{grad} f(x_k))$ by Algorithm 1; ▷ # $2n^3 + d$
4: **while** $\|\mathrm{gf}_k^d\| > \epsilon$ **do**
5: Obtain $\eta_k \in \mathbb{R}^d$, intrinsic representation of a tangent vector $\eta^w \in \mathrm{T}_{x_k}\mathcal{M}$, by the following algorithm, Step 6 to Step 16:
6: $q \leftarrow \mathrm{gf}_k^d$;
7: **for** $i = k-1, k-2, \ldots, k-l$ **do** ▷ # $l(\lambda_m^d + 2d)$
8: $\xi_i \leftarrow \rho_i q^T s_i$;
9: $q \leftarrow q - \xi_i y_i$;
10: **end for**
11: $r \leftarrow \gamma_k q$; ▷ # d
12: **for** $i = k-l, k-l+1, \ldots, k-1$ **do** ▷ # $l(\lambda_m^d + 2d)$
13: $\omega \leftarrow \rho_i r^T y_i$;
14: $r \leftarrow r + s_i(\xi_i - \omega)$;
15: **end for**
16: set $\eta_k = -r$, $\alpha_k = 1$; ▷ # d
17: Compute $\eta_k^w = D2E_{x_k}(\eta_k)$ by Algorithm 2; ▷ # $2n^3 + n(n+1)/2$
18: If $$\|\operatorname{grad} \mathrm{gf}_k^d\| / \|\mathrm{gf}_0^d\| < accuracy,$$
then set $x_{k+1} = R_{x_k}(\alpha_k \eta_k^w)$ and go to Step 22; ▷ # λ_r^d
19: Compute $\tilde{x}_k = R_{x_k}(\alpha_k \eta_k^w)$; ▷ # λ_r^d
20: If $$f(\tilde{x}_k) \le f(x_k) + \delta \alpha_k \eta_k^T \mathrm{gf}_k^d,$$
then set $x_{k+1} = \tilde{x}_k$ and go to Step 22;
21: Set $\alpha_k = \varrho \alpha_k$ and go to Step 19;
22: Compute $x_{k+1} = L_{k+1} L_{k+1}^T$, $\operatorname{grad} f(x_{k+1})$;
23: Compute $\mathrm{gf}_k^d = E2D_{x_k}(\operatorname{grad} f(x_k))$ by Algorithm 1; ▷ # $2n^3 + d$
24: Define $s_k = \alpha_k \eta_k$ and $y_k = \mathrm{gf}_{k+1}^d - \mathrm{gf}_k^d$;[7] ▷ # $2d$
25: Compute $a = y_k^T s_k$ and $b = \|s_k\|_2^2$; ▷ # $2\lambda_m^d$

if $\frac{a}{b} \geq 10^{-4}\|\mathrm{gf}_k^d\|_2$ **then** ▷ # λ_m^d

  Compute $c = \|y_k^{(k+1)}\|_2^2$ and define $\rho_k = 1/a$ and $\gamma_{k+1} = a/c$; ▷ # λ_m^d

  Add s_k, y_k and ρ_k into storage and if $l \geq m$, then discard vector pair $\{s_{k-l}, y_{k-l}\}$ and scalar ρ_{k-l} from storage, else $l \leftarrow l + 1$;

else

  Set $\gamma_{k+1} \leftarrow \gamma_k$, $\{\rho_k, \ldots, \rho_{k-l+1}\} \leftarrow \{\rho_{k-1}, \ldots, \rho_{k-l}\}$, $\{s_k, \ldots, s_{k-l+1}\} \leftarrow \{s_{k-1}, \ldots, s_{k-l}\}$ and $\{y_k, \ldots, y_{k-l+1}\} \leftarrow \{y_{k-1}, \ldots, y_{k-l}\}$

end if

$k = k + 1$;

end while

CHAPTER 3

DIVERGENCE FUNCTIONS ON $\mathcal{S}_{++}^{\mathrm{n}}$ AND DIVERGENCE-BASED MEANS

3.1 Introduction

As discussed in Chapter 2, the geodesic distance induced by the affine-invariant metric (2.1.1) provides a natural dissimilarity measure between two SPD matrices

$$\delta_R(X,Y) = \|\log(L^{-1}YL^{-T})\|_{\mathrm{F}}, \tag{3.1.1}$$

where $X, Y \in \mathcal{S}_{++}^{\mathrm{n}}$ and $X = LL^T$. The Karcher mean based on δ_R provides an attractive way of averaging a collection of SPD matrices as it satisfies all the desired geometric properties in the ALM list [6]. However, the computational cost of the geodesic distance (3.1.1) and the Karcher mean associated with it increases dramatically with the dimension of the manifold (i.e., the size of the SPD matrices). As shown in Section 2.4.3, the dominant computation time for the Karcher mean computation using LRBFGS is in the problem-related operations, i.e., function and gradient evaluations. Our implementation of manifold- and algorithm-related objects is the cheapest we can get so far. This motivates us to consider using divergences as alternatives to the geodesic distance (3.1.1).

A divergence is similar to a distance except that it does not need to satisfy the triangle inequality nor symmetry, which also provides a measure of dissimilarity between two elements. In fact, in recent years, the matrix divergences have begun to draw more and more attention due to its simplicity, efficiency and robustness to outliers, e.g., see [5,8,21,26,27,75,89,91]. The idea of using divergences to define the mean of a collection of SPD matrices has been studied in literature [22,25, 71,72,84,85]. The divergence-based mean is defined in a similar way as the Karcher mean, which also requires solving an optimization problem on $\mathcal{S}_{++}^{\mathrm{n}}$. Unlike the Karcher mean computation that is extensively tackled by Riemannian optimization methods, the divergence-based mean is mainly computed by fixed point algorithms when the closed form solution is not available. This chapter is devoted to various divergences on $\mathcal{S}_{++}^{\mathrm{n}}$ and the computation of divergence-based means.

The main contributions of this chapter are:

- We provide a review of commonly used divergences on $\mathcal{S}_{++}^{\mathrm{n}}$ and divergence-based means for a set of SPD matrices.
- We generalize the proof for the geodesic convexity of the log-determinant α-divergence from the case of $\alpha = 0$ in [84] to the general case of $-1 < \alpha < 1$.
- We apply our Riemannian optimization techniques on $\mathcal{S}_{++}^{\mathrm{n}}$ developed in Chapter 2 to compute the divergence-based mean when the closed form expression is not available, which outperforms the state-of-the-art fixed point algorithm.
- For the SPD log-determinant α-divergence-based mean computation, we cast the state-of-the-art fixed point algorithm into a Riemannian steepest descent for a choice of the cost function, retraction, and stepsize strategy.
- We use the problem of computing the SPD log-determinant α-divergence-based mean to empirically illustrate the relationship between the BB stepsizes and the eigenvalues of the Riemannian Hessian of the cost function. We consider different versions of BB stepsizes, including BB1, BB2, and $\mathrm{ABB}_{\mathrm{min}}$.

3.2 The α-divergence from Jensen convexity gap

3.2.1 Preliminaries and definitions

Let $\varphi : \Omega \to \mathbb{R}$ be a strictly convex and differentiable real-valued function defined on a closed convex set $\Omega \subset \mathbb{R}^m$. A one parameter family of skewed divergences [22, 74] generated by φ, called the Jensen divergence, is defined as

$$\delta_{\varphi,\lambda}^2(x, y) = \frac{1}{\lambda(1-\lambda)}[(1-\lambda)\varphi(x) + \lambda\varphi(y) - \varphi((1-\lambda)x + \lambda y)] \tag{3.2.1}$$

$$= \frac{1}{\lambda(1-\lambda)}[(\varphi(x)\varphi(y))_\lambda - \varphi((xy)_\lambda)], \tag{3.2.2}$$

where $x, y \in \Omega$, $\lambda \in (0, 1)$, and $(ab)_\lambda = (1-\lambda)a + \lambda b$ denotes the linear interpolant between a and b. A geometrical illustration of the Jensen divergence for scalars is given in Figure 3.1. As depicted in Figure 3.1, the Jensen divergence is proportional to the vertical distance between the point $((xy)_\lambda, (\varphi(x)\varphi(y))_\lambda)$ lying on the line segment connecting points $(x, \varphi(x))$ and $(y, \varphi(y))$ and the point $((xy)_\lambda, \varphi((xy)_\lambda))$ lying on the curve of φ.

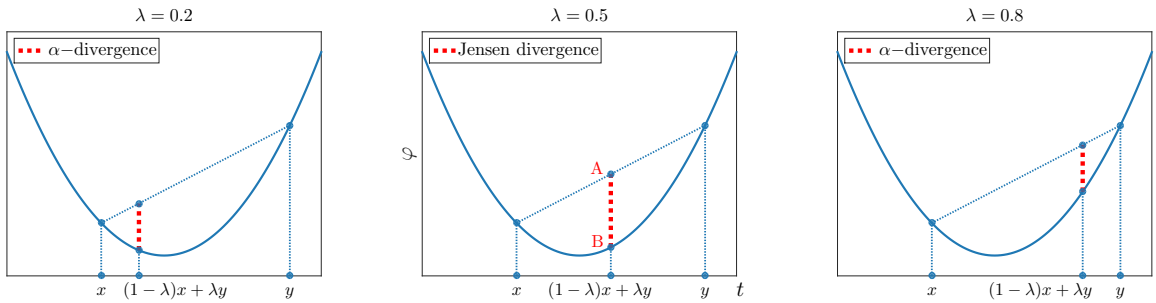


Figure 3.1: Geometrical illustration of the skewed Jensen divergence.

With the change of parameter $\lambda = (1+\alpha)/2$, the skewed Jensen divergence (3.2.1) can be written as

$$\delta^2_{\varphi,\alpha}(x,y) = \frac{4}{1-\alpha^2}[\frac{1-\alpha}{2}\varphi(x) + \frac{1+\alpha}{2}\varphi(y) - \varphi(\frac{1-\alpha}{2}x + \frac{1+\alpha}{2}y)], \tag{3.2.3}$$

where $\alpha \in (-1,1)$. The α-version (3.2.3) of the Jensen divergence is called the α-divergence family in [99]. The α-version is preferred since this version possess a dual symmetry with respect to the change $\alpha \to -\alpha$.

For values $\alpha = 1$ and $\alpha = -1$, the α-divergence is defined by taking limit as $\alpha \to 1$ and $\alpha \to -1$. As $\alpha \to 1$, we have

$$\begin{aligned}
\delta^2_{\varphi,1}(x,y) &= \lim_{\alpha\to 1} \delta^2_{\varphi,\alpha}(x,y) && (3.2.4)\\
&= \lim_{\alpha\to 1} \frac{2}{1+\alpha}[\varphi(x) - \varphi(y) - \frac{2}{1-\alpha}(\varphi(y + \frac{1-\alpha}{2}(x-y)) - \varphi(y))] && (3.2.5)\\
&= \varphi(x) - \varphi(y) - \lim_{\alpha\to 1} \frac{\varphi(y + t(x-y)) - \varphi(y)}{t} && (3.2.6)\\
&= \varphi(x) - \varphi(y) - \mathrm{D}\,\varphi(y)[x-y] && (3.2.7)\\
&= \varphi(x) - \varphi(y) - \langle \nabla\varphi(y), x-y\rangle. && (3.2.8)
\end{aligned}$$

Note that equation (3.2.8) is actually the Bregman divergence defined in [20], denoted by $\delta^2_{\varphi,\mathrm{B}}(x,y)$. Similarly, when $\alpha \to -1$, we have $\delta^2_{\varphi,-1}(x,y) = \delta^2_{\varphi,\mathrm{B}}(y,x)$. Throughout this dissertation, we follow the tradition and call (3.2.8) the Bregman divergence. A geometrical illustration of the Bregman divergence is given in Figure 3.2. It measures the vertical distance between the point $(x,\varphi(x))$ and the tangent at $(y,\varphi(y))$.

Both the α-divergence (3.2.3) and the Bregman divergence (3.2.8) can be naturally extended to $\mathcal{S}^{\mathrm{n}}_{++}$, e.g., see [22,71,74]. Given a strictly convex and differentiable real-valued function $\phi : \mathcal{S}^{\mathrm{n}}_{++} \to \mathbb{R}$

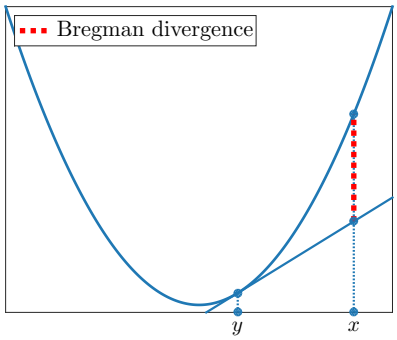


Figure 3.2: Geometrical illustration of the Bregman divergence.

and $X, Y \in \mathcal{S}^{\mathrm{n}}_{++}$, the α-divergence with $-1 < \alpha < 1$ is defined as

$$\delta^2_{\phi,\alpha}(X,Y) = \frac{4}{1-\alpha^2}[\frac{1-\alpha}{2}\phi(X) + \frac{1+\alpha}{2}\phi(Y) - \phi(\frac{1-\alpha}{2}X + \frac{1+\alpha}{2}Y)]. \tag{3.2.9}$$

The Bregman divergence, denoted by $\delta^2_{\phi,\mathrm{B}}$, is defined as

$$\delta^2_{\phi,\mathrm{B}}((X,Y) = \phi(X) - \phi(Y) - \langle \nabla\phi(Y), X - Y\rangle, \tag{3.2.10}$$

where $\langle X, Y\rangle = \mathrm{tr}(XY)$. Different choices of ϕ give different divergences. Commonly used convex functions on $\mathcal{S}^{\mathrm{n}}_{++}$ are [74]:

- quadratic entropy:
$$\phi(X) = \mathrm{tr}(X^T X), \tag{3.2.11}$$
- log-determinant (also called Burg) entropy:
$$\phi(X) = -\log\det X, \tag{3.2.12}$$
- von Neumann entropy:
$$\phi(X) = \mathrm{tr}(X\log X - X). \tag{3.2.13}$$

In this chapter, we focus on the log-determinant entropy (3.2.12) and the von Neumann entropy (3.2.13).

3.2.2 Symmetrized divergence

A divergence is asymmetric in general. There are two common ways to symmetrize a divergence [27]:

- Type 1:
$$\delta^2_{\mathrm{S}\phi}(X,Y) = \frac{1}{2}(\delta^2_\phi(X,Y) + \delta^2_\phi(Y,X)), \tag{3.2.14}$$

- Type 2:
$$\delta^2_{\mathrm{S}\phi}(X,Y) = \frac{1}{2}(\delta^2_\phi(X, \frac{X+Y}{2}) + \delta^2_\phi(Y, \frac{X+Y}{2})). \tag{3.2.15}$$

3.2.3 The LogDet α-divergence

When the associated function $\phi(X)$ in (3.2.9) is the log-determinant function (3.2.12), we get the log-determinant α-divergence [22]. Below gives the formal definition.

Definition 3.2.1. *For $-1 < \alpha < 1$, the family of log-determinant (hereafter abbreviated LD or LogDet) α-divergence functions on $\mathcal{S}^{\mathrm{n}}_{++}$ is defined as*

$$\delta^2_{\mathrm{LD},\alpha}(X,Y) = \frac{4}{1-\alpha^2}\log\frac{\det(\frac{1-\alpha}{2}X + \frac{1+\alpha}{2}Y)}{\det(X)^{\frac{1-\alpha}{2}}\det(Y)^{\frac{1+\alpha}{2}}}. \tag{3.2.16}$$

Remark 3.2.1. *With the change of parameter $p = \frac{1-\alpha}{2}$, the LogDet α-divergence can be written as*

$$\delta^2_{\mathrm{LD},p}(X,Y) = \frac{1}{p(1-p)}\log\frac{\det(pX + (1-p)Y)}{\det X^p \det Y^{1-p}}, \quad 0 < p < 1. \tag{3.2.17}$$

The most frequently mentioned advantage of the LogDet α-divergence (3.2.16) against the geodesic distance δ_{R} is its computational efficiency. The computation of (3.2.16) requires three Cholesky factorizations (for $\frac{1-\alpha}{2}X + \frac{1+\alpha}{2}Y$, X, and Y), while computing the geodesic distance involves eigenvalue decomposition. We will see later that the computational advantage of the LogDet α-divergence is more impressive when one wants to compute the related means. In addition, the LogDet α-divergence enjoys several desired invariance properties [22]:

1. Invariance under congruence transformations
$$\delta^2_{\mathrm{LD},\alpha}(SAS^T, SBS^T) = \delta^2_{\mathrm{LD},\alpha}(A,B) \text{ for any invertible } S. \tag{3.2.18}$$

2. Dual-invariance under inversion
$$\delta^2_{\mathrm{LD},\alpha}(A^{-1},B^{-1}) = \delta^2_{\mathrm{LD},-\alpha}(A,B). \tag{3.2.19}$$

3. Dual symmetry
$$\delta^2_{\mathrm{LD},\alpha}(A,B) = \delta^2_{\mathrm{LD},-\alpha}(B,A). \tag{3.2.20}$$

The LogDet α-divergence (3.2.16) is asymmetric except for $\alpha = 0$. But it can be symmetrized using (3.2.14) and (3.2.15), and two symmetric forms of the LogDet α-divergence are expressed as

$$\delta^2_{\mathrm{S1LD},\alpha}(X,Y) = \frac{2}{1-\alpha^2}\log\frac{\det(\frac{1-\alpha}{2}X+\frac{1+\alpha}{2}Y)(\frac{1-\alpha}{2}Y+\frac{1+\alpha}{2}X)}{\det(XY)}, \tag{3.2.21}$$

and

$$\delta^2_{\mathrm{S2LD},\alpha}(X,Y) = \frac{2}{1-\alpha^2}\log\frac{\det(\frac{3-\alpha}{4}X+\frac{1+\alpha}{4}Y)(\frac{3-\alpha}{4}Y+\frac{1+\alpha}{4}X)}{\det(XY)^{\frac{1-\alpha}{2}}\det(\frac{X+Y}{2})^{1+\alpha}}. \tag{3.2.22}$$

Remark 3.2.2. *$\delta^2_{\mathrm{LD},0}$ is also called the Stein divergence, and fully studied in [84, 85]. It is shown in [84] that $\delta^2_{\mathrm{LD},0}$ is the square of a distance function (i.e., $\delta_{\mathrm{LD},0}$ is a distance function), and it shares several common geometric properties with the geodesic distance δ^2_R, see [84, Table 4.1].*

3.2.4 The LogDet Bregman divergence

Recall that the Bregman divergence on $\mathcal{S}^{\mathrm{n}}_{++}$, denoted by $\delta^2_{\phi,\mathrm{B}}$, is defined as

$$\delta^2_{\phi,\mathrm{B}}(X,Y) = \phi(X) - \phi(Y) - \langle\nabla\phi(Y), X-Y\rangle, \tag{3.2.23}$$

where $X, Y \in \mathcal{S}^{\mathrm{n}}_{++}$ and $\langle X,Y\rangle = \mathrm{tr}(XY)$. The LogDet Bregman divergence is defined using $\phi(X) = -\log\det X$, and is given as

$$\delta^2_{\mathrm{LD,B}}(X,Y) = \mathrm{tr}(Y^{-1}X - I) - \log\det(Y^{-1}X). \tag{3.2.24}$$

The LogDet Bregman divergence is also called the Kullback-Leibler divergence in [72]. It is easy to verify that the LogDet Bregman divergence is invariant under congruence transformations. In addition, the LogDet Bregman divergence is asymmetric, but it can be easily symmetrized using (3.2.14) and (3.2.15). Two symmetric versions of the LogDet Bregman divergence are expressed as

$$\delta^2_{\mathrm{S1LD,B}}(X,Y) = \frac{1}{2}\mathrm{tr}(Y^{-1}X + X^{-1}Y - 2I), \tag{3.2.25}$$

and

$$\delta^2_{\mathrm{S2LD,B}}(X,Y) = \log\det(\frac{X+Y}{2}) - \frac{1}{2}\log\det(XY). \tag{3.2.26}$$

Notice that (3.2.26) coincides with the LogDet α-divergence with $\alpha = 0$. The type 1 symmetrized LogDet Bregman divergence (3.2.25) is also called the Jeffrey divergence (or J-divergence) in [42,91]. We can verify that both (3.2.25) and (3.2.26) are invariant under congruence and inversion.

3.2.5 The von Neumann α-divergence

The von Neumann function $\phi(X) = \operatorname{tr}(X \log X - X)$ arises in quantum mechanics [76], whose domain is the set of positive semidefinite matrices by using the convention that $0 \log 0 = 0$. The von Neumann α-divergence is defined as

$$\delta^2_{\mathrm{VN},\alpha}(X,Y) = \frac{4}{1-\alpha^2} \operatorname{tr}\{\frac{1-\alpha}{2} X \log X + \frac{1+\alpha}{2} Y \log Y - (\frac{1-\alpha}{2} X + \frac{1+\alpha}{2} Y) \log(\frac{1-\alpha}{2} X + \frac{1+\alpha}{2} Y)\}. \tag{3.2.27}$$

From (3.2.27), we can verify that the von Neumann α-divergence satisfies the following invariance properties:

1. Invariance under rotations

$$\delta^2_{\mathrm{VN},\alpha}(OXO^T, OYO^T) = \delta^2_{\mathrm{VN},\alpha}(X,Y) \text{ for any } O \in \mathrm{SO}(n). \tag{3.2.28}$$

2. Dual symmetry

$$\delta^2_{\mathrm{VN},\alpha}(X,Y) = \delta^2_{\mathrm{VN},-\alpha}(Y,X). \tag{3.2.29}$$

It is clear from the dual symmetry that the von Neumann divergence is asymmetric except for $\alpha = 0$, which is given by

$$\delta^2_{\mathrm{VN},0}(X,Y) = 4 \operatorname{tr}\{\frac{1}{2} X \log X + \frac{1}{2} Y \log Y - (\frac{X+Y}{2}) \log(\frac{X+Y}{2})\}. \tag{3.2.30}$$

We note that the computation of the von Neumann α-divergence (3.2.27) requires 3 eigenvalue decompositions, which makes it more expensive than the computation of the geodesic distance δ_R, the LogDet α-divergence $\delta^2_{\mathrm{LD},\alpha}$, and the LogDet Bregman divergence $\delta^2_{\mathrm{LD,B}}$. The advantage of the von Neumann α-divergence is that it is defined over positive semidefinite matrices. But in this dissertation, we focus on the symmetric positive definite matrices.

3.2.6 The von Neumann Bregman divergence

The von Neumann Bregman divergence [74], denoted by $\delta^2_{\mathrm{VN,B}}$, is defined using $\phi(X) = \operatorname{tr}(X \log X - X)$ for the Bregman divergence (3.2.23)

$$\delta^2_{\mathrm{VN,B}}(X,Y) = \operatorname{tr}(X(\log X - \log Y) - X + Y). \tag{3.2.31}$$

Note that (3.2.31) is referred to as the von Neumann divergence in [30, 59, 74] and the quantum relative entropy in [76]. We keep the word "Bregman" just to emphasize that it is a Bregman

divergence other than a α-divergence from Jensen convexity gap. The von Neumann Bregman divergence (3.2.31) is invariant under rotations, and its computation requires two eigenvalue decompositions. It is shown in [30] that (3.2.31) is finite if and only if the range of Y contains the range of X, i.e., range$(X) \subseteq$ range(Y). For this reason, the von Neumann Bregman divergence is often used in low-rank matrix nearness problems, e.g., see [30, 59, 60].

The von Neumann Bregman divergence is asymmetric, and its symmetrized versions are given by

$$\delta^2_{\mathrm{S1VN,B}}(X,Y) = \frac{1}{2}\operatorname{tr}(X\log X + Y\log Y - X\log Y - Y\log X) \tag{3.2.32}$$

$$= \frac{1}{2}\operatorname{tr}(X(\log X - \log Y) + Y(\log Y - \log X)), \tag{3.2.33}$$

and

$$\delta^2_{\mathrm{S2VN,B}}(X,Y) = \operatorname{tr}(\frac{1}{2}X\log X + \frac{1}{2}Y\log Y - (\frac{X+Y}{2})\log(\frac{X+Y}{2})). \tag{3.2.34}$$

Since $\delta^2_{\mathrm{S1VN,B}}(X,Y)$ is obtained using $(\delta^2_{\mathrm{VN,B}}(X,Y) + \delta^2_{\mathrm{VN,B}}(Y,X))/2$, it is finite if and only if range(X) = range(Y). That is, the type 1 symmetrized von Neumann Bregman divergence $\delta^2_{\mathrm{S1VN,B}}(X,Y)$ enjoys a range-space preserving property, which appears to be important for the analysis of rank deficient matrices. But in this dissertation, we only consider the positive definite matrices. In addition, we note that the symmetrized von Neumann Bregman divergence (3.2.34) coincides with the von Neumann α-divergence with $\alpha = 0$, i.e., equation (3.2.30).

3.3 Sided and symmetrized means based on the divergence

3.3.1 Definitions

Given a divergence function on $\mathcal{S}^{\mathrm{n}}_{++}$, one can define the mean of a collection of SPD matrices $\{A_1, \ldots, A_K\}$ in a similar way as the Karcher mean. Due to the asymmetry of divergence functions, the notion of right mean and left mean are used. The right mean and left mean coincide if the divergence is symmetric.

Definition 3.3.1. *The right mean of a collection of SPD matrices $\{A_1, \ldots, A_K\}$ associated with divergence function $\delta^2_\phi(x,y)$ is defined as the minimizer to the sum of divergence*

$$\mu^r = \underset{X\in\mathcal{S}^{\mathrm{n}}_{++}}{\arg\min}\, f(X), \quad \textit{with } f : \mathcal{S}^{\mathrm{n}}_{++} \to \mathbb{R},\ X \mapsto \sum_{i=1}^{K} \delta^2_\phi(A_i, X). \tag{3.3.1}$$

Definition 3.3.2. *The left mean of a collection of SPD matrices* $\{A_1, \ldots, A_K\}$ *associated with divergence function* $\delta_\phi^2(x, y)$ *is defined as the minimizer to the sum of divergence*

$$\mu^l = \underset{X \in \mathcal{S}_{++}^{\mathrm{n}}}{\arg\min} f(X), \quad \textit{with } f : \mathcal{S}_{++}^{\mathrm{n}} \to \mathbb{R},\ X \mapsto \sum_{i=1}^{K} \delta_\phi^2(X, A_i). \tag{3.3.2}$$

3.3.2 Means based on the LogDet α-divergence

The right mean based on the LogDet α-divergence for a set of SPD matrices, $\{A_1, \ldots, A_K\} \in \mathcal{S}_{++}^{\mathrm{n}}$, is defined as

$$\mu^r = \underset{X \in \mathcal{S}_{++}^{\mathrm{n}}}{\arg\min} f(X), \quad \text{with } f : \mathcal{S}_{++}^{\mathrm{n}} \to \mathbb{R},\ X \mapsto \sum_{i=1}^{K} \delta_{\mathrm{LD},\alpha}^2(A_i, X), \tag{3.3.3}$$

where $\delta_{\mathrm{LD},\alpha}^2$ is given in (3.2.16). This problem has been studied in [22], and they claimed that the right mean μ^r is given by the unique solution to the necessary optimality condition

$$\mathrm{grad}^{\mathrm{euc}} f(X) = 0 \iff X^{-1} = \frac{1}{K} \sum_{i=1}^{K} (\frac{1-\alpha}{2} X + \frac{1+\alpha}{2} A_i)^{-1}, \tag{3.3.4}$$

where $\mathrm{grad}^{\mathrm{euc}} f(X)$ denotes the gradient of f under the Euclidean metric (1.5.7). The existence of the unique solution to matrix equation (3.3.4) is proved in [22]. The necessary condition (3.3.4) is also sufficient since the cost function $f(X) \to \infty$ as $X \to 0$ and $X \to \infty$. That is, optimization problem (3.3.3) has a unique minimizer. [84] analyzed the mean problem (3.3.3) for $\alpha = 0$, and they proved that $\delta_{\mathrm{LD},0}^2$ is jointly geodesically convex under the affine-invariant metric $g_X(\xi, \eta) = \mathrm{tr}(\xi X^{-1} \eta X^{-1})$ where $\xi, \eta \in \mathrm{T}_X \mathcal{S}_{++}^{\mathrm{n}}$. That is, $\delta_{\mathrm{LD},0}^2$ satisfies the following inequality

$$\delta_{\mathrm{LD},0}^2(X_1 \#_t X_2, Y_1 \#_t Y_2) \leq (1-t)\delta_{\mathrm{LD},0}^2(X_1, Y_1) + t\delta_{\mathrm{LD},0}^2(X_2, Y_2), \tag{3.3.5}$$

where $A \#_t B := \gamma(t) = A^{1/2}(A^{-1/2} B A^{-1/2})^t A^{1/2}$ for $t \in [0, 1]$ denotes the geodesic between A and B under the affine-invariant metric with $\gamma(0) = A$ and $\gamma(1) = B$. Here we generalize their proof for the geodesic convexity to the general LogDet α-divergence (3.2.16). Before proving Theorem 3.3.1, we recall some useful results. For simplicity of notation, we drop the subscript when $t = 1/2$, i.e., $A \# B = A \#_{1/2} B$ denotes the Karcher mean of A and B.

Proposition 3.3.1. *(Joint-concavity [57]) Let* $A, B, C, D \in \mathcal{S}_{++}^{\mathrm{n}}$. *Then*

$$(A \# B) + (C \# D) \leq (A + C) \# (B + D). \tag{3.3.6}$$

Proposition 3.3.2. *(Consistency with scalars) Let $A, B \in \mathcal{S}^{\mathrm{n}}_{++}$ and $p \in \mathbb{R}_+$. Then*

$$p(A\#B) = (pA)\#(pB). \tag{3.3.7}$$

Proposition 3.3.3. *Let $A, B \in \mathcal{S}^{\mathrm{n}}_{++}$. Then*

$$\log\det(A\#B) = \frac{1}{2}(\log\det A + \log\det B). \tag{3.3.8}$$

Proof. From the determinant identity, we have

$$\det(A\#B) = (\det A \det B)^{1/2}.$$

Take the log of both sides, we have

$$\log\det(A\#B) = \frac{1}{2}(\log\det A + \log\det B). \tag{3.3.9}$$

□

Theorem 3.3.1. *The function $\delta^2_{\mathrm{LD},\alpha}(X, Y)$ is jointly geodesically convex, i.e., for any $X_1, X_2, Y_1, Y_2 \in \mathcal{S}^{\mathrm{n}}_{++}$ and $t \in (0, 1)$, we have*

$$\delta^2_{\mathrm{LD},\alpha}(X_1\#_t X_2, Y_1\#_t Y_2) \leq (1-t)\delta^2_{\mathrm{LD},\alpha}(X_1, Y_1) + t\delta^2_{\mathrm{LD},\alpha}(X_2, Y_2). \tag{3.3.10}$$

Proof. To prove the theorem, we use the p-version of the LogDet α-divergence given in (3.2.17). Since $\delta^2_{\mathrm{LD},p}$ is continuous, it suffices to show that for any $X_1, X_2, Y_1, Y_2 \in \mathcal{S}^{\mathrm{n}}_{++}$,

$$\delta^2_{\mathrm{LD},p}(X_1\#X_2, Y_1\#Y_2) \leq \frac{1}{2}\delta^2_{\mathrm{LD},p}(X_1, Y_1) + \frac{1}{2}\delta^2_{\mathrm{LD},p}(X_2, Y_2). \tag{3.3.11}$$

From Proposition 3.3.2 and Proposition 3.3.1, we have

$$p(X_1\#X_2) + (1-p)(Y_1\#Y_2) = (pX_1)\#(pX_2) + ((1-p)Y_1)\#((1-p)Y_2) \tag{3.3.12}$$

$$\leq (pX_1 + (1-p)Y_1)\#(pX_2 + (1-p)Y_2). \tag{3.3.13}$$

Since the log-determinant function is monotonic, then we have

$$\log\det(p(X_1\#X_2) + (1-p)(Y_1\#Y_2)) \leq \log\det((pX_1 + (1-p)Y_1)\#(pX_2 + (1-p)Y_2)) \tag{3.3.14}$$

$$= \frac{1}{2}\log\det(pX_1 + (1-p)Y_1) + \frac{1}{2}\log\det(pX_2 + (1-p)Y_2). \tag{3.3.15}$$

Combining the above inequality with the identity

$$- \log\det(X_1\#X_2)^p - \log\det(Y_1\#Y_2)^{1-p} \tag{3.3.16}$$

$$= -p\log\det(X_1\#X_2) - (1-p)\log\det(Y_1\#Y_2) \tag{3.3.17}$$

$$= -\frac{p}{2}(\log\det X_1 + \log\det X_2) - \frac{1-p}{2}(\log\det Y_1 + \log\det Y_2) \tag{3.3.18}$$

$$= -\frac{1}{2}\log\det X_1^p Y_1^{1-p} - \frac{1}{2}\log\det X_2^p Y_2^{1-p}. \tag{3.3.19}$$

Then we get

$$\log\frac{\det(p(X_1\#X_2)+(1-p)(Y_1\#Y_2))}{\det(X_1\#X_2)^p(Y_1\#Y_2)^{1-p}} \le \frac{1}{2}\log\frac{\det(pX_1+(1-p)Y_1)}{\det(X_1^pY_1^{1-p})} + \frac{1}{2}\log\frac{\det(pX_2+(1-p)Y_2)}{\det(X_2^pY_2^{1-p})}. \tag{3.3.20}$$

That is,

$$\delta^2_{\mathrm{LD},p}(X_1\#X_2, Y_1\#Y_2) \le \frac{1}{2}\delta^2_{\mathrm{LD},p}(X_1,Y_1) + \frac{1}{2}\delta^2_{\mathrm{LD},p}(X_2,Y_2). \tag{3.3.21}$$

□

From Theorem 3.3.1, the cost function in the right mean problem (3.3.3) is geodesically convex. Hence a local minimum point is also a global minimum point. However, a closed-form solution for problem (3.3.3) is unknown in general, except for $K = 2$. Unlike the Karcher mean computation that is extensively tackled by Riemannian optimization methods, the LogDet α-divergence based mean is often computed by fixed point algorithms, see [22, 74]. An Euclidean Newton's method is considered in [22] which, however, fails to converge in some of their numerical experiments. [22] studied the special case of $\alpha = 0$ and gave a fixed point algorithm to compute the divergence-based mean with a convergence study. [25, 26, 84, 85] applied this fixed point algorithm to compute the divergence-based mean. We propose to solve the sided mean problem (3.3.3) using implementation techniques developed in Section 2.3 and Riemannian optimization algorithms, including RSD, LRBFGS, RBFGS, and RNewton. In addition, we could explain the fixed point algorithm in the literature in Riemannian optimization framework.

Next, we present the problem-related operations, i.e., function and gradient evaluations, for problem (3.3.3). The cost function in problem (3.3.3) is given by

$$f(X) = \frac{4}{1-\alpha^2}\sum_{i=1}^{K}\log\frac{\det(\frac{1-\alpha}{2}A_i + \frac{1+\alpha}{2}X)}{\det(A_i)^{\frac{1-\alpha}{2}}\det(X)^{\frac{1+\alpha}{2}}} \tag{3.3.22}$$

$$= \frac{4}{1-\alpha^2}\sum_{i=1}^{K}\{\log\det(\frac{1-\alpha}{2}A_i + \frac{1+\alpha}{2}X) - \frac{1+\alpha}{2}\log\det X - \frac{1-\alpha}{2}\log\det A_i\} \tag{3.3.23}$$

$$= \frac{4}{1-\alpha^2}\sum_{i=1}^{K}\{\operatorname{tr}\log(\frac{1-\alpha}{2}A_i + \frac{1+\alpha}{2}X) - \frac{1+\alpha}{2}\operatorname{tr}\log X - \frac{1-\alpha}{2}\operatorname{tr}\log A_i\}. \tag{3.3.24}$$

From (3.3.23) to (3.3.24), we use identity $\log\det A = \operatorname{tr}\log A$. Our cost function evaluation is based on formula (3.3.23), which requires only two Cholesky factorizations. Notice that the $\log\det A_i$ term is a constant, and thus can be ignored. The number of flops required by function evaluation (3.3.23) is $2Kn^3/3 + o(Kn^3)$.

The directional derivative of the cost function (3.3.23) at x in the direction of η is given in [22]

$$\mathrm{D}\, f(X)[\eta] = \frac{2}{1-\alpha}\operatorname{tr}\sum_{i=1}^{K}\{(\frac{1-\alpha}{2}A_i + \frac{1+\alpha}{2}X)^{-1} - X^{-1}\}\eta. \tag{3.3.25}$$

Thus the Riemannian gradient of the cost function under the affine-invariant metric (2.1.1) is given by

$$\operatorname{grad} f(X) = \frac{2}{1-\alpha}\sum_{i=1}^{K}\{X(\frac{1-\alpha}{2}A_i + \frac{1+\alpha}{2}X)^{-1}X - X\}. \tag{3.3.26}$$

The evaluation of (3.3.26) requires the computation of the inverse of $(\frac{1-\alpha}{2}A_i + \frac{1+\alpha}{2}X)$, whose Cholesky factorization is available from the cost function evaluation. So evaluating the Riemannian gradient needs extra $3Kn^3 + o(Kn^3)$ flops. We compare the computational complexities of the problem-related operations for the Karcher mean computation and the LogDet α-divergence based mean computation in Table 3.1. The efficiency of the latter is one of the key reasons for using it as an alternative to the Karcher mean.

The Riemannian Newton's method requires the action of Hessian. Under the affine-invariant metric (2.1.1), the action of Riemannian Hessian of the cost function f is given by

$$\operatorname{Hess} f(X)[\eta] = \mathrm{D}(\operatorname{grad} f(X))[\eta] - \frac{1}{2}(\eta X^{-1}\operatorname{grad} f(X) + \operatorname{grad} f(X)X^{-1}\eta). \tag{3.3.27}$$

For simplicity of notation, we let $B_i = (\frac{1-\alpha}{2}A_i + \frac{1+\alpha}{2}X)^{-1}$. Then we have

$$\mathrm{D}(\operatorname{grad} f(X))[\eta] = \frac{2}{1-\alpha}\sum_{i=1}^{K}\{\eta B_i X + X B_i \eta - X B_i(\frac{1+\alpha}{2})\eta B_i X - \eta\}. \tag{3.3.28}$$

Table 3.1: The complexities of the problem-related operations for the computation of means based on the Riemannian geodesic distance δ_R (3.1.1) and the LogDet α-divergence $\delta^2_{\mathrm{LD},\alpha}$ (3.2.16).

	cost function	Riemannian gradient
mean based on $\delta^2_{\mathrm{LD},\alpha}$	$2Kn^3/3$	$3Kn^3$
mean based on δ^2_R	$18Kn^3$	$5Kn^3$

Thus, the action of Hessian is given by

$$\operatorname{Hess} f(X)[\eta] = \frac{1}{1-\alpha}\sum_{i=1}^{K}\{(\eta B_i X + X B_i \eta) - (1+\alpha) X B_i \eta B_i X\}. \tag{3.3.29}$$

Next, we will show that the fixed point algorithm in [22, 74] coincides with the Riemannian steepest descent method for a choice of cost function, retraction, and stepsize strategy. The fixed point algorithm in [22, Algorithm 1] is derived from the necessary optimality condition (3.3.4), which is given by

$$X_{k+1} = K(\sum_{i=1}^{K}(\frac{1-\alpha}{2}A_i + \frac{1+\alpha}{2}X_k)^{-1})^{-1}. \tag{3.3.30}$$

With the change of variable $Y = X^{-1}$, an equivalent expression of (3.3.30) is given as [22, Algorithm 1']

$$Y_{k+1} = \frac{1}{K}\sum_{i=1}^{K}(\frac{1-\alpha}{2}A_i + \frac{1+\alpha}{2}Y_k^{-1})^{-1}. \tag{3.3.31}$$

It is claimed in [22, Proposition 3.13] that $\{Y_k\}$ monotonically converges to the inverse of the LogDet α-divergence-based right mean with the initializations of the arithmetic mean and the harmonic-arithmetic mean. We propose the following cost function g

$$g(Y) = \delta^2_{\mathrm{LD},\alpha}(A_i, Y^{-1}) \tag{3.3.32}$$

$$= \frac{4}{1-\alpha^2}\sum_{i=1}^{K}\{\log\det(\frac{1-\alpha}{2}A_i + \frac{1+\alpha}{2}Y^{-1}) + \frac{1+\alpha}{2}\log\det Y - \frac{1-\alpha}{2}\log\det A_i\}. \tag{3.3.33}$$

The Riemannian gradient of the cost function $g(Y)$ under the affine-invariant metric (2.1.1) is given by

$$\operatorname{grad} g(Y) = \frac{2}{1-\alpha}\sum_{i=1}^{K}\{Y - (\frac{1-\alpha}{2}A_i + \frac{1+\alpha}{2}Y^{-1})^{-1}\}. \tag{3.3.34}$$

Combining equation (3.3.31) and (3.3.34), we have

$$Y_{k+1} = \frac{1}{K}\{KY_k - \frac{1-\alpha}{2}\operatorname{grad} g(Y_k)\} \tag{3.3.35}$$

$$= Y_k - \frac{1-\alpha}{2K}\operatorname{grad} g(Y_k). \tag{3.3.36}$$

Equation (3.3.36) implies that the fixed point procedure given in (3.3.31) is a Riemannian steepest descent method using the Euclidean retraction $R_X(\eta_X) = X + \eta_X$ and a constant stepsize $(1-\alpha)/2K$.

We end this section by a quick glance at the symmetric mean based on the symmetrized LogDet α-divergence, which is defined as

$$\mu = \underset{X\in\mathcal{S}^{\mathrm{n}}_{++}}{\arg\min} f(X), \quad \text{with } f : \mathcal{S}^{\mathrm{n}}_{++} \to \mathbb{R},\ X \mapsto \sum_{i=1}^{K} \delta^2_{\mathrm{S1LD},\alpha}(A_i, X). \tag{3.3.37}$$

It is easy to verify that the symmetric mean defined in (3.3.37) is invariant under congruence transformation and inversion. The cost function in (3.3.37) is given by

$$f(X) = \frac{2}{1-\alpha^2}\sum_{i=1}^{K}\{\log\det(\frac{1-\alpha}{2}A_i+\frac{1+\alpha}{2}X)+\log\det(\frac{1-\alpha}{2}X+\frac{1+\alpha}{2}A_i)\}-\log\det X-\log\det A_i\}. \tag{3.3.38}$$

Since $\delta^2_{\mathrm{LD},\alpha}$ is jointly geodesically convex, the cost function f is geodesically convex, and thus a local minimum is also global. The Riemannian gradient of (3.3.38) is given by

$$\operatorname{grad} f(X) = \frac{2}{1-\alpha^2}\sum_{i=1}^{K}\{\frac{1+\alpha}{2}X(\frac{1-\alpha}{2}A_i+\frac{1+\alpha}{2}X)^{-1}X+\frac{1-\alpha}{2}X(\frac{1-\alpha}{2}X+\frac{1+\alpha}{2}A_i)^{-1}X-X\}. \tag{3.3.39}$$

The computation of the cost function (3.3.38) requires $Kn^3 + O(Kn^3)$ flops, and its Riemannian gradient (3.3.39) needs $6Kn^3 + O(Kn^3)$ flops.

3.3.3 Means based on the LogDet Bregman divergence

Means based on the LogDet Bregman divergence have been thoroughly studied in [72, Lemma 17.4.3], and they turn out to have closed-form expressions.

Lemma 3.3.1. *Let $\{A_1, \ldots, A_K\}$ be a collection of SPD matrices, $\mathcal{A}(A_1, \ldots, A_K) = \frac{1}{K}\sum_{i=1}^{K} A_i$ be their arithmetic mean, $\mathcal{H}(A_1, \ldots, A_K) = K(\sum_{i=1}^{K} A_i^{-1})^{-1}$ be their harmonic mean, and $\mathcal{G}(A_1, \ldots, A_K)$ be their geometric mean.*

1. *The right mean based on* $\delta^2_{\mathrm{LD,B}}$ (3.2.24) *is given by the arithmetic mean, i.e.,*

$$\mathcal{A}(A_1,\ldots,A_K) = \underset{X\in\mathcal{S}^{\mathrm{n}}_{++}}{\arg\min} \sum_{i=1}^{K} \delta^2_{\mathrm{LD,B}}(A_i, X). \tag{3.3.40}$$

2. *The left mean based on* $\delta^2_{\mathrm{LD,B}}$ (3.2.24) *is given by the harmonic mean, i.e.,*

$$\mathcal{H}(A_1,\ldots,A_K) = \underset{X\in\mathcal{S}^{\mathrm{n}}_{++}}{\arg\min} \sum_{i=1}^{K} \delta^2_{\mathrm{LD,B}}(X, A_i). \tag{3.3.41}$$

3. *The symmetric mean based on* $\delta^2_{\mathrm{S1LD,B}}$ (3.2.25) *is given by the geometric mean of the arithmetic mean and the harmonic mean, i.e.,*

$$\mathcal{G}(\mathcal{A}(A_1,\ldots,A_K),\mathcal{H}(A_1,\ldots,A_K)) = \underset{X\in\mathcal{S}^{\mathrm{n}}_{++}}{\arg\min} \sum_{i=1}^{K} \delta^2_{\mathrm{S1LD,B}}(A_i, X). \tag{3.3.42}$$

3.3.4 Means based on the von Neumann Bregman divergence

Given a collection of SPD matrices $\{A_1,\ldots,A_K\} \in \mathcal{S}^{\mathrm{n}}_{++}$, the right mean μ^r and left mean μ^l associated with the von Neumann Bregman divergence are given by, respectively,

$$\mu^r = \underset{X\in\mathcal{S}^{\mathrm{n}}_{++}}{\arg\min}\, \delta^2_{\mathrm{VN,B}}(A_i, X) = \underset{X\in\mathcal{S}^{\mathrm{n}}_{++}}{\arg\min} \sum_{i=1}^{K} \mathrm{tr}(A_i \log A_i - A_i \log X - A_i + X) \tag{3.3.43}$$

and

$$\mu^l = \underset{X\in\mathcal{S}^{\mathrm{n}}_{++}}{\arg\min}\, \delta^2_{\mathrm{VN,B}}(X, A_i) = \underset{X\in\mathcal{S}^{\mathrm{n}}_{++}}{\arg\min} \sum_{i=1}^{K} \mathrm{tr}(X \log X - X \log A_i - X + A_i). \tag{3.3.44}$$

Note that we can omit the constant terms for the cost functions in problem (3.3.43) and (3.3.44). A difficulty in tackling problem (3.3.43) is due to the fact that an analytic form for the gradient of $\mathrm{tr}\, A_i \log X$ with respect to X is not known. The Euclidean gradient of $\mathrm{tr}(X \log X - X)$ is shown to be $\log X$ in [88]. Thus, the Riemannian gradient of the cost function in (3.3.44) is given by

$$\mathrm{grad}\, f(X) = \sum_{i=1}^{K} X(\log X - \log A_i)X. \tag{3.3.45}$$

The solution, denoted by μ^l, to problem (3.3.44) must satisfy the optimality condition

$$\mathrm{grad}\, f(\mu^l) = 0 \Leftrightarrow \sum_{i=1}^{K} (\log \mu^l - \log A_i) = 0. \tag{3.3.46}$$

In other words,

$$\mu^l = \exp(\frac{1}{K}\sum_{i=1}^{K}\log A_i). \tag{3.3.47}$$

The left mean based on the von Neumann Bregman divergence coincides with the Log-Euclidean Frechet mean in [10].

The computation of the symmetric mean based on the von Neumann Bregman divergence encounters the same difficulty as the right mean, since it also requires the gradient of $\operatorname{tr} A_i \log X$.

3.4 Numerical experiments I: computation of the LogDet α-divergence-based means

In this section, we compare the performances of the state-of-the-art fixed point algorithm (FP) proposed in [22, Algorithm 1] with a number of Riemannian optimization algorithms, including Riemannian steepest descent (RSD), LRBFGS described in Algorithm 7, RBFGS in [51, Algorithm 2], and Riemannian Newton's method (RNewton) using a truncated conjugate gradient method in [2].

In our practical implementation, RSD and LRBFGS are combined with the Armijo backtracking line search with Armijo parameter $\delta = 10^{-4}$ and backtracking reduction factor $\varrho = 0.5$. RBFGS and RNewton are combined with the Wolfe line search with parameters $c_1 = 10^{-4}$ and $c_2 = 0.999$. The initial stepsize for RSD is taken as the classical strategy in [93, (3.44)]

$$\alpha_{k+1} = \min\{1, 1.01 \cdot \frac{2(f_{k+1} - f_k)}{g_{x_{k+1}}(gf_{k+1}, -gf_{k+1})}\}, \tag{3.4.1}$$

where $f_k = f(x_k)$ and $gf_k = \operatorname{grad} f(x_k)$. We set the maximum stepsize $\alpha_{max} = 100$ and the minimum stepsize α_{min} is machine epsilon. The initial stepsize in the first iteration is set to be 1. Unless otherwise specified, our choice of the initial iterate is the harmonic-arithmetic mean of data matrices. The maximum number of iterations is set to be 500. For LRBFGS, we use the BB2 initial scaling method with $m_a = 5$, and test different memory sizes m as specified in the legends of figures. Recall that LRBFGS with $m = 0$ is actually a steepest descent method with a BB stepsize. We run the algorithms until they reach their highest accuracy.

For simplicity of notation, throughout this section we denote the number, dimension, and condition number of the matrices by K, n, and κ respectively. For each choice of (K, n) and the range of conditioning desired, a single experiment comprises generating 50 different sets of K

random $n \times n$ matrices with appropriate condition numbers, and running all the algorithms on each set with identical parameters. For each dataset, we compute the true solution using LRBFGS with $m = 2$ with a high accuracy such that $\|\operatorname{grad} f(X_k)\|/\|\operatorname{grad} f(X_0)\| < 10^{-12}$. The result of the experiment is the distance to the true solution averaged over the 50 sets as a function of iteration and time. To obtain sufficiently stable timing results, an average time is taken of several runs for a total runtime of at least 1 minute.

All experiments are carried out using C++, compiled with gcc-4.7.x., and performed on the Florida State University HPC system using Quad-Core AMD Opteron(tm) Processor 2356 2.3GHz.

Figure 3.3 and 3.4 report the results of tests conducted on data sets with $K = 100$ and $n = 3$. The condition number of the data matrices is between 10 and 10^6. Figure 3.3 presents a zoom-in of Figure 3.4. Various values of α are tested as specified at the top of each plot. Note that we use the same datasets for different values of α. We observe that the value of α has a significant impact on the performance of FP. For nonnegative α, i.e., $\alpha = 0$, 0.5 and 0.9, the considered Riemannian optimization algorithms impressively outperform the FP. The performance of FP becomes better as the value of α decreases, but is still outperformed by the considered Riemannian optimization algorithms. For the considered Riemannian optimization algorithms, RNewton requires the least number of iterations to achieve a desired accuracy, but this advantage is nullified by its high computational cost per iteration. LRBFGS with $m = 2$ is clearly the winner in terms of time efficiency for all the values of α tested except for $\alpha = -0.9$. When $\alpha = -0.9$, the Riemannian optimization algorithms tend to struggle in the early steps, but get better in the later phase. If a high accuracy is desired, LRBFGS is clearly the choice of method. For some values of α, RBFGS performs similarly as LRBFGS with $m = 2$, while its performance depends on the value of α. However, as the dimension of the manifold increases, RBFGS becomes computationally demanding. We also note that RBB, i.e., LRBFGS with $m = 0$, performs better than RSD in terms of number of iterations and time efficiency. This implies that a suitable choice of the initial stepsize can accelerate the steepest descent method. In our opinion, the acceleration of RBB is due to the fact that the BB stepsize injects some second order information into the stepsizes, while the classical stepsize (3.4.1) only uses the first order information. We will explore more about the BB methods in the next section.

3.5 Numerical experiments II: the BB stepsizes and Hessian eigenvalues

In this section, we investigate the relationship between the BB stepsizes, including BB1, BB2, and $\text{ABB}_{\min}$, and the eigenvalues of the Riemannian Hessian of the cost function. It is seen from Section 3.4 that the appropriate choice of the initial stepsize could accelerate the convergence speed of the steepest descent methods. The BB stepsizes make use of the second order information of the cost function, while the classical NW's stepsize given in (3.4.1) only uses the first order information. For readers' convenience, we summarize the formulas for different stepsizes:

- NW's: $\alpha_{k+1} = \min\{1, 1.01 \cdot 2(f_{k+1} - f_k)/g(gf_{k+1}, -gf_{k+1})\}$,
- BB1: $\alpha_{k+1} = g(s_k, s_k)/g(s_k, y_k)$,
- BB2: $\alpha_{k+1} = g(s_k, y_k)/g(y_k, y_k)$,
- $\text{ABB}_{\min}$: $\alpha_{k+1} = \min\{\alpha_j^{\text{BB2}} : j = \max(1, k - m_a), \ldots, k\}$ if $\alpha_{k+1}^{\text{BB}_2}/\alpha_{k+1}^{\text{BB1}} < \tau$; otherwise $\alpha_{k+1} = \alpha_{k+1}^{\text{BB1}}$.

In order to illustrate the practical behavior of the BB stepsizes, we analyze the numerical results obtained by solving the LogDet α-divergence-based right mean problem (3.3.3). The objective function is

$$f(X) = \frac{4}{1-\alpha^2} \sum_{i=1}^{K} \{\log\det(\frac{1-\alpha}{2} A_i + \frac{1+\alpha}{2} X) - \frac{1+\alpha}{2} \log\det X\}. \tag{3.5.1}$$

The intrinsic representation of the Riemannian Hessian of f at x is given by

$$H = \frac{1}{1-\alpha} \sum_{i=1}^{K} O^T \{(L^T B_i L) \otimes I_n + I_n \otimes (L^T B_i L) - (1+\alpha)(L^T B_i L) \otimes (L^T B_i L)\} O, \tag{3.5.2}$$

where $B_i = (\frac{1-\alpha}{2} A_i + \frac{1+\alpha}{2} X)^{-1}$, $X = LL^T$, and O is the orthonormal basis under the Euclidean metric (1.5.7), i.e., $O = \{\text{vec}(e_i e_i^T) : i = 1, \ldots, n\} \bigcup \{\text{vec}(\frac{1}{\sqrt{2}}(e_i e_j^T + e_j e_i^T)),\ i < j,\ i = 1, \ldots, n,\ j = 1, \ldots, n\} \in \mathbb{R}^{n^2 \times d}$ with $\{e_1, \ldots, e_n\}$ being the standard basis of $\mathbb{R}^n$. Note that the dimension of the Hessian matrix H is $d \times d$, where $d = n(n+1)/2$. We compute all the eigenvalues of H using the eig function in MATLAB.

In Figure 3.5-3.10, we compare the practical behavior of BB1, BB2, $\text{ABB}_{\min}$, and LRBFGS. The aim of the comparison is two fold: (1) to analyze the relationship between the stepsizes α_k generated by different BB methods and the eigenvalues of the current Riemannian Hessian; (2) to

compare the convergence speed and time efficiency of different methods. In each figure, plots in the first row display the values of the reciprocal of three BB stepsizes as specified in the title and the eigenvalues of the Riemannian Hessian of f as a function of iteration. In the plots, the green mark 'x' indicates the reciprocal of the initial stepsize generated by each BB method, the blue mark 'o' indicates the the reciprocal of the final stepsize that satisfies the Armijo condition, the pink dot depicts the eigenvalues of current Riemannian Hessian and the dotted lines are obtained by linear interpolation. The plots in the second row of each figure show, respectively, the eigenvalues of the Riemannian Hessian of each iteration obtained by LRBFGS, and evolution of distance between current iterate and the exact solution with respect to iterations and time. For LRBFGS, we use the BB2 initial scaling method with memory size $m = 2$. Since the initial stepsize for LRBFGS is always 1, we only display the values of eigenvalues of the Riemannian Hessian for each iteration. All the results shown in Figure 3.5-3.10 are obtained from the same dataset with varying values of α, which are representatives of multiple experiments. Note that α without subscript is the parameter in the LogDet α-divergence. The dataset consists 200 SPD matrices of size 6×6 and condition number $1 \leq \kappa \leq 10^7$. For each value of α, we test two different iterates: the arithmetic-harmonic mean and a random initial iterate that is far away from the true solution. For ABB_{min}, the parameters are chosen on the basis of the literature [31]. We set $m_a = 5$ and $\tau = 0.7$.

We first observe that LRBFGS wins over RBB in all cases in terms of convergence speed and computation time. The sequences $\{1/\alpha_k\}$ generated by three BB stepsizes appear to follow the distribution of the eigenvalues of the Riemannian Hessian of the objective function. BB2 method performs better than BB1 in general, especially when $\alpha \geq 0$ and the random initial iterate is far away from the true solution. When the arithmetic-harmonic mean is used as the initial iterate, which is a good initial start-point, we see from Figure 3.5, 3.7 and 3.9 that the eigenvalues of the Riemannian Hessian of f converge to those at the minimizer quickly. When the random initial iterate is used, from Figure 3.6, 3.8 and 3.10, we see that the eigenvalues of the Riemannian Hessian associated with the BB1 method converge more quickly than those associated with the BB2 method. We also observe that the sequences $1/\alpha_k$ generated by ABB_{min} formula tend to take groups of large values, interleaved with some smaller values. This implies that the ABB_{min} method tend to take groups of small stepsizes, interleaved with some large stepsizes.

3.6 Summary and comparison

We have discussed various dissimilarity measures on $\mathcal{S}^{\mathrm{n}}_{++}$ and related means for a set of SPD matrices. Table 3.2 lists the desired invariance properties of a measure for SPD matrices. Table 3.3 and Table 3.4 summarize the definitions and properties of various distances/divergences mentioned in the chapter. Table 3.5 summarizes the literatures dealing with the divergence-based matrix means and algorithms used to compute the means.

Table 3.2: A summary of desired invariance properties.

Properties	Formulas
Scaling invariance	$\delta(sX, sY) = \delta(X, Y)$, $\forall s \in \mathbb{R}^+$
Rotation invariance	$\delta(OXO^T, OYO^T) = \delta(X, Y)$, $\forall O \in \mathrm{SO}(n)$
Congruence invariance	$\delta(SXS^T, SYS^T) = \delta(X, Y)$, $\forall S \in \mathrm{GL}(n)$
Inversion invariance	$\delta(X, Y) = \delta(X^{-1}, Y^{-1})$

Remark 3.6.1.

- *The Stein divergence is also known as Jensen-Bregman LogDet divergence, and it belongs to the LogDet α-divergence with $\alpha = 0$, i.e., $\delta_{\mathrm{S}} = \delta_{\mathrm{LD},0}$.*
- *The LogDet Bregman divergence is also referred to as the Kullback-Leibler divergence [72], and can be obtained from the LogDet α-divergence by letting $\alpha \to 1$, i.e., $\delta_{\mathrm{LD,B}} = \delta_{\mathrm{LD},1}$.*
- *The Jeffrey divergence is obtained from the LogDet Bregman divergence by symmetrization, i.e.,*
$$\delta^2_{\mathrm{J}}(X, Y) = \frac{1}{2}(\delta_{\mathrm{LD,B}}(X, Y) + \delta_{\mathrm{LD,B}}(Y, X)). \tag{3.6.1}$$
- *The triangle inequality of the square root of the Stein divergence, i.e., δ_{S} is shown in [84].*

3.7 Conclusions

In this chapter, we provide a review of commonly used divergences on $\mathcal{S}^{\mathrm{n}}_{++}$ and divergence-based means for a set of SPD matrices. We focus on the computation of the log-determinant α-divergence-based mean. We show that the log-determinant α-divergence is jointly geodesically

Table 3.3: Notation and definitions of the distances/divergences on $\mathcal{S}^{\mathrm{n}}_{++}$.

Distance & $\sqrt{\text{Divergence}}$	Notation	Formula	Symmetric	Triangle inequality
Euclidean distance	δ_{E}	$\|X - Y\|_{\mathrm{F}}$	Yes	Yes
Cholesky-Frobenius [32]	δ_{CL}	$\|\mathrm{Chol}(X) - \mathrm{Chol}(Y)\|_{\mathrm{F}}$	Yes	Yes
Affine-invariant distance [78]	δ_{R}	$\|\log(X^{-1/2} Y X^{-1/2})\|_{\mathrm{F}}$	Yes	Yes
Log-Euclidean distance [10]	δ_{LE}	$\|\log(X) - \log(Y)\|_{\mathrm{F}}$	Yes	Yes
LogDet α-divergence [22]	$\delta_{\mathrm{LD},\alpha}$	$\sqrt{\frac{4}{1-\alpha^2} \log \frac{\det(\frac{1-\alpha}{2} A_i + \frac{1+\alpha}{2} X)}{\det(A_i)^{\frac{1-\alpha}{2}} \det(X)^{\frac{1+\alpha}{2}}}}$	No	No
Stein divergence [22, 84]	δ_{S}	$2\sqrt{\log\det(\frac{X+Y}{2}) - \frac{1}{2}\log\det(XY)}$	Yes	Yes
LogDet Bregman divergence [22, 60]	$\delta_{\mathrm{LD,B}}$	$\sqrt{\mathrm{tr}(XY^{-1}) - \log\det(XY^{-1}) - n}$	No	No
Jeffrey divergence [91]	δ_{J}	$\sqrt{\frac{1}{2}\mathrm{tr}(XY^{-1} + YX^{-1}) - n}$	Yes	No
von Neumann α-divergence	$\delta_{\mathrm{VN},\alpha}$	see (3.2.27)	No	No
von Neumann Bregman divergence [74]	$\delta_{\mathrm{VN,B}}$	$\sqrt{\mathrm{tr}(X(\log X - \log Y) - X + Y)}$	No	No

convex. We apply our Riemannian optimization techniques to handle this computational task, which outperforms the state-of-the-art method. In addition, we cast the state-of-the-art method into Riemannian optimization framework. Moreover, we provide a numerical illustration of the relationship between the Barzilai-Borwein stepsizes and the eigenvalues of the Riemannian Hessian of the cost function.

Table 3.4: Distances/divergences on $\mathcal{S}^{\mathrm{n}}_{++}$ and their invariance properties.

Distance & Divergence	Scaling invariance	Rotation invariance	Congruence invariance	Inversion invariance
Euclidean distance	No	Yes	No	No
Cholesky-Frobenius	No	No	No	No
Affine-invariant distance	Yes	Yes	Yes	Yes
Log-Euclidean distance	Yes	Yes	No	Yes
LogDet α-divergence	Yes	Yes	Yes	No
Stein divergence	Yes	Yes	Yes	Yes
LogDet Bregman divergence	Yes	Yes	Yes	No
Jeffrey divergence	Yes	Yes	Yes	Yes
von Neumann α-divergence	No	Yes	No	No
von Neumann Bregman divergence	No	Yes	No	No

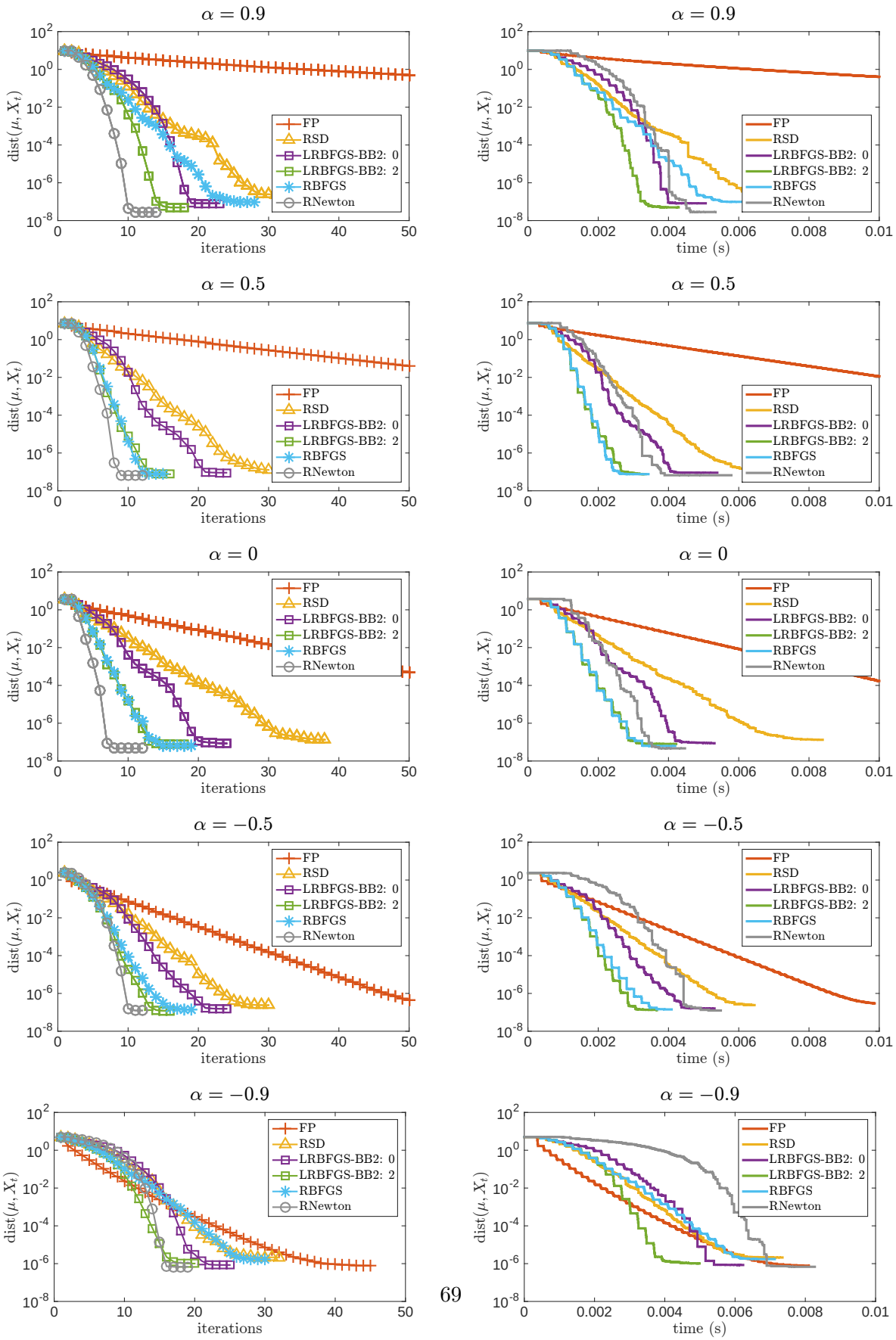


Figure 3.3: Comparison of different algorithms with $K = 100$, $n = 3$, and $10 \leq \kappa(A_i) \leq 10^6$.

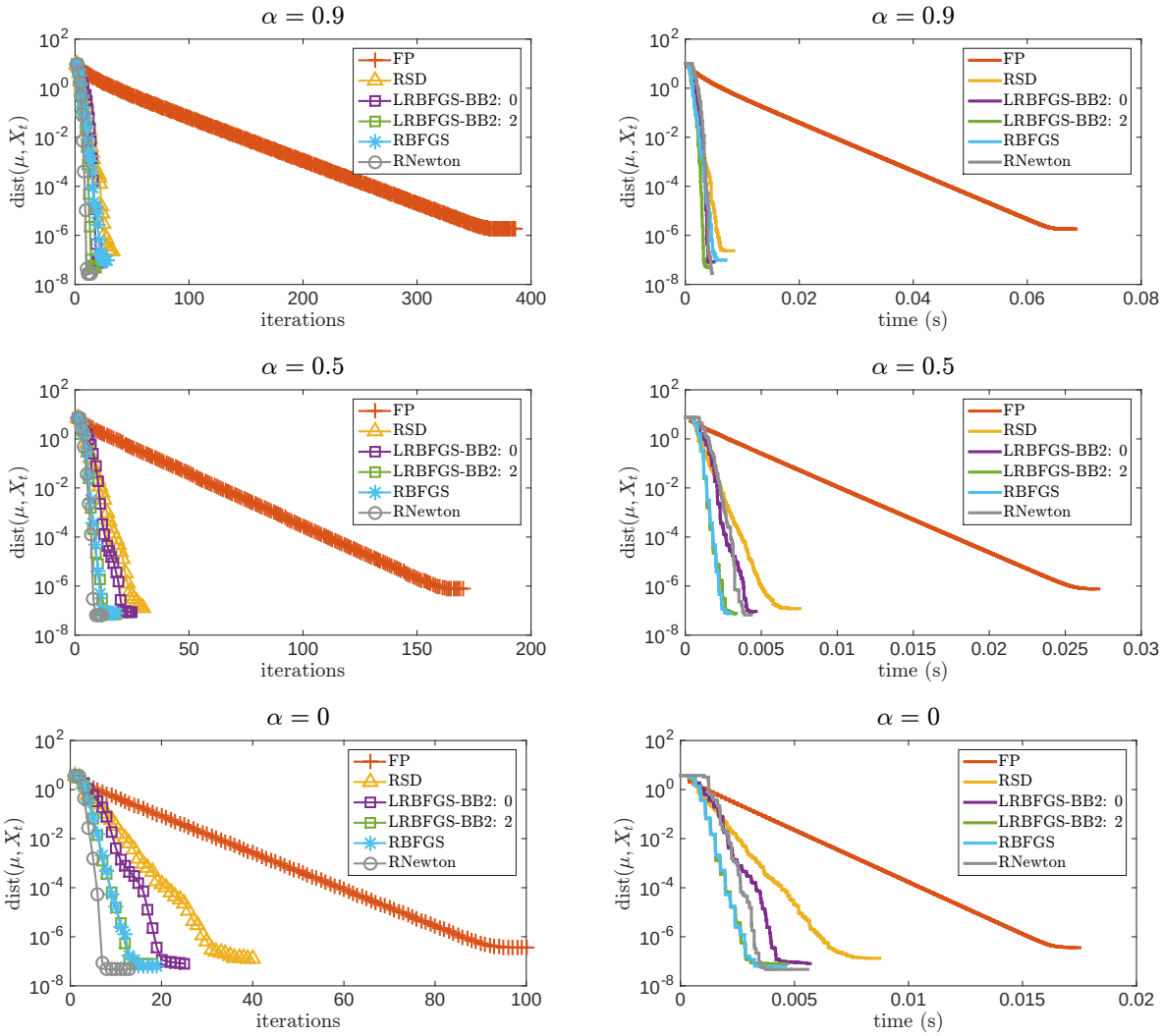


Figure 3.4: Comparison of different algorithms with $K = 100$, $n = 3$, and $10 \leq \kappa(A_i) \leq 10^6$.

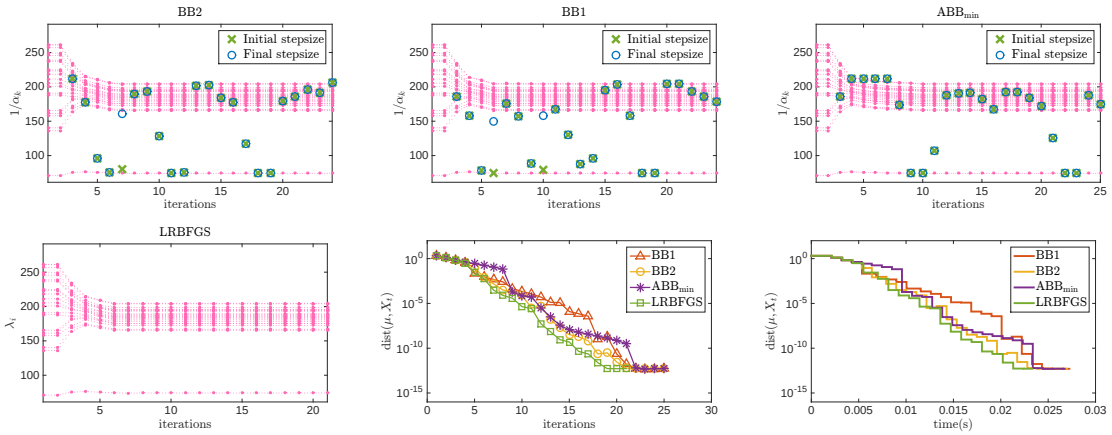


Figure 3.5: $\alpha = -0.5$. The initial iterate is the arithmetic-harmonic mean.

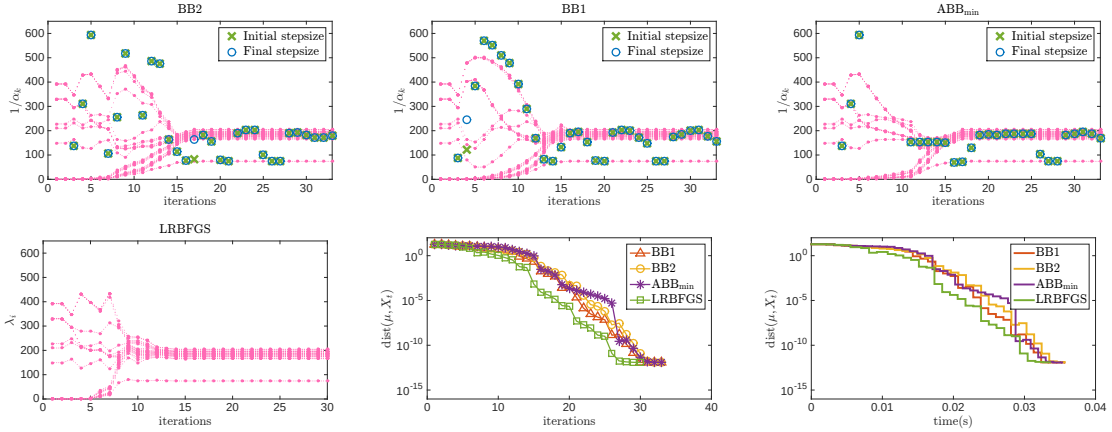


Figure 3.6: $\alpha = -0.5$. The initial iterate is randomly generated.

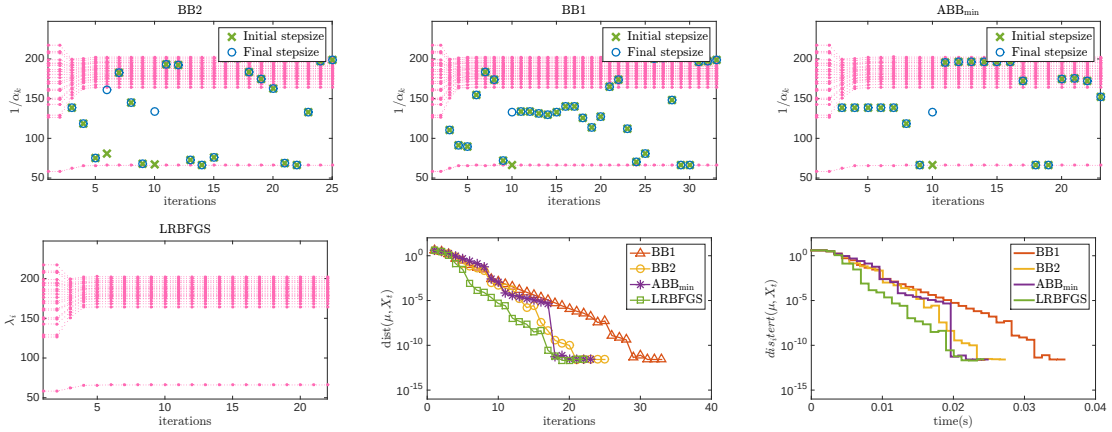


Figure 3.7: $\alpha = 0$. The initial iterate is the arithmetic-harmonic mean.

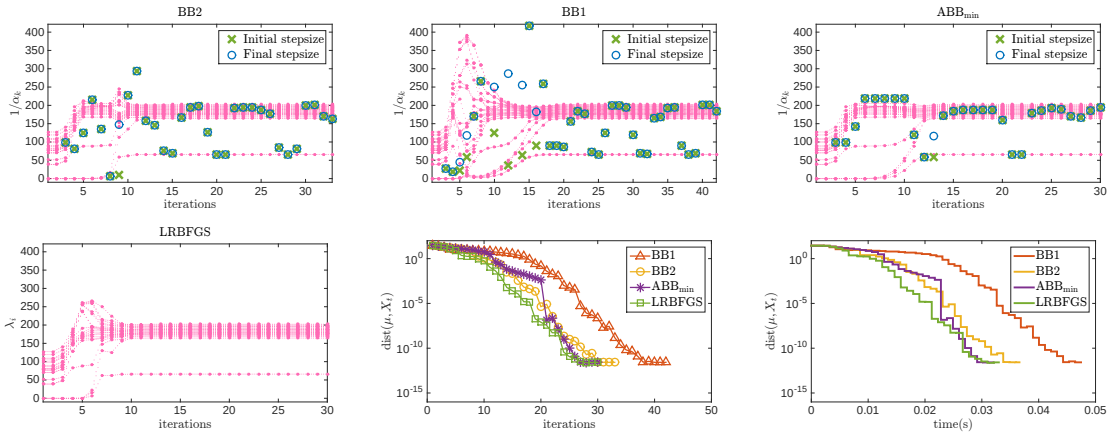


Figure 3.8: $\alpha = 0$. The initial iterate is randomly generated.

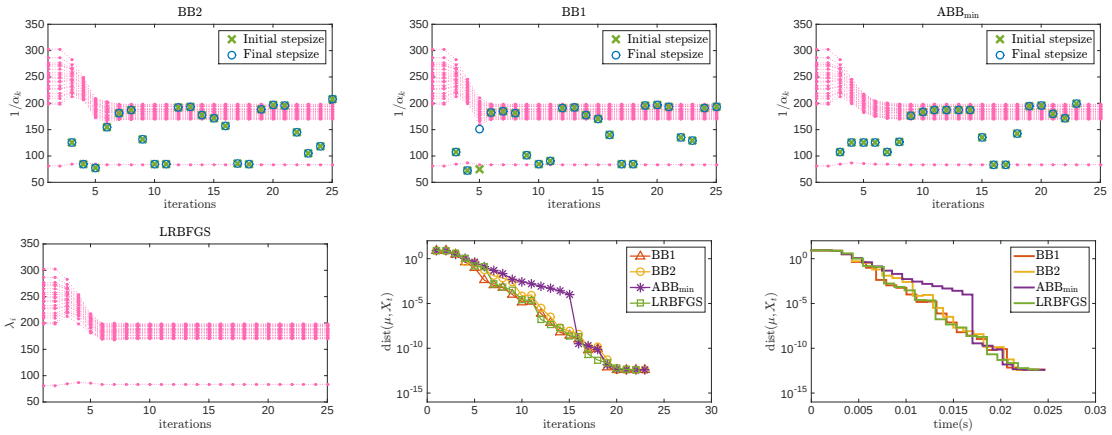


Figure 3.9: $\alpha = 0.5$. The initial iterate is the arithmetic-harmonic mean.

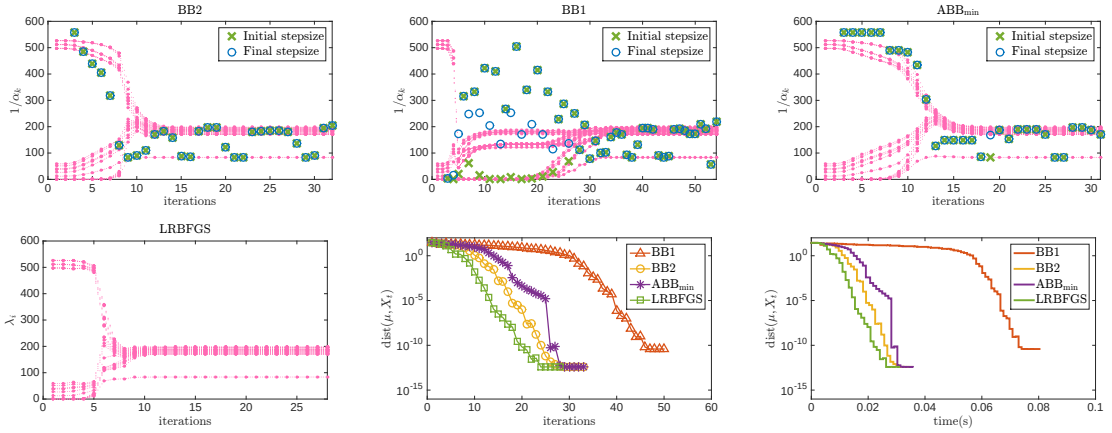


Figure 3.10: $\alpha = 0.5$. The initial iterate is randomly generated.

Table 3.5: A summary of the literatures dealing with the divergence-based matrix means and algorithms used to compute the means.

Distance & Divergence	Reference	Closed-form expression/Algorithms
Euclidean distance	—	• arithmetic mean
Cholesky-Frobenius	—	—
Affine-invariant distance	[71, 72]	• fixed point iteration • Riemannian optimization: RSD, RCG, LRBFGS, RBFGS, RNewton
Log-Euclidean distance	[10]	• log-Euclidean mean: $\exp(\sum_{i=1}^{K} \log(A_i)/K)$
LogDet α-divergence	[22, 74]	• fixed point iteration (which is the same as the concave-convex procedure [97]) • Euclidean Newton's method
Stein divergence	[22, 25, 26, 84, 85]	• fixed point iteration
LogDet Bregman divergence	[71, 72]	• right mean: arithmetic mean • left mean: harmonic mean
Jeffrey divergence	[71, 72]	• arithmetic-harmonic mean
von Neumann α-divergence	—	—
von Neumann Bregman divergence	—	• right mean: — • left mean: log-Euclidean mean

CHAPTER 4

L^1 RIEMANNIAN MEDIAN COMPUTATION ON $\mathcal{S}^n_{++}$

4.1 Introduction

In the Euclidean space, it is known that the median is preferred to the mean in the presence of outliers due to the robustness of the former and the sensitivity of the latter. This is illustrated in Figure 4.1, where the mean is dragged towards the outliers lying at the top right corner, while the median appears to be a better estimator of centrality. It is shown in [65] that half of the points needs to be corrupted in order to corrupt the median.

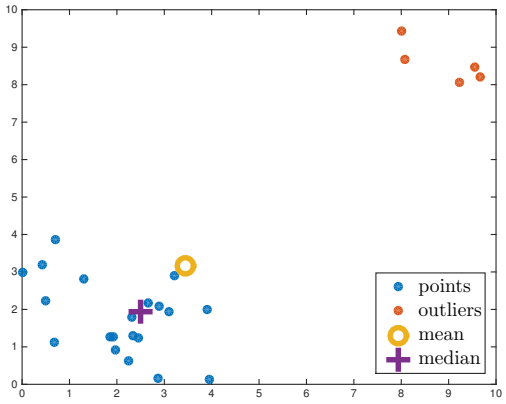


Figure 4.1: The geometric mean and median in $\mathbb{R}^2$ space.

Given a set of points $\{a_1, \ldots, a_K\} \in \mathbb{R}^n$, with the usual Euclidean distance $\|\cdot\|$, the geometric median is defined as the point $m \in \mathbb{R}^n$ minimizing the sum of distance $f(x) = \sum_{i=1}^K \|x - a_i\|$. The geometric median can be computed by an iterative algorithm introduced by Weiszfeld [92], which is essentially an Euclidean steepest descent. The gradient of f exists when $x \in \mathbb{R}^n$ not equal to any a_i and is given by

$$\nabla f(x) = \sum_{i=1}^{K} \frac{1}{\|x - a_i\|}(x - a_i). \tag{4.1.1}$$

Later Ostresh [77] improved Weiszfeld's algorithm and proposed an update iteration as

$$x_{k+1} = x_k - \alpha G_k, \quad G_k = \sum_{i=1}^{K} \frac{a_i}{\|x_k - a_i\|} \Big(\sum_{i=1}^{K} \frac{1}{\|x_k - a_i\|}\Big)^{-1}, \tag{4.1.2}$$

where $\alpha > 0$ is a stepsize. Notice if current iterate coincides with one of the data points, i.e., $x_k = a_j$, then $G_k = a_j$. It was proven in [77] that the iteration (4.1.2) converges to the unique median for $\alpha \leq 2$ and when the data points are not all colinear. Figure 4.2 illustrates the median and mean of 3 points in $\mathbb{R}^2$. In the third plot, the median coincides with one of the data points.

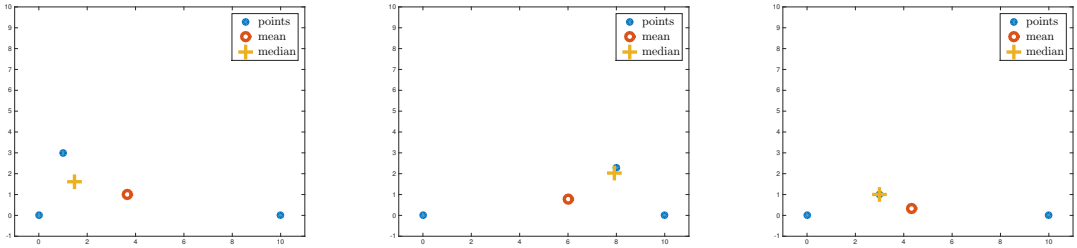


Figure 4.2: The geometric median and mean for 3 points in $\mathbb{R}^2$ space.

This notion of geometric median can be extended to the $\mathcal{S}^{\mathrm{n}}_{++}$ manifold. Given a set of SPD matrices $\{A_1, \ldots, A_K\}$, their Riemannian median is defined as the minimizer to the sum of distances

$$\mu^{md} = \underset{X \in \mathcal{S}^{\mathrm{n}}_{++}}{\arg\min} \sum_{i=1}^{K} \delta(A_i, X), \tag{4.1.3}$$

where $\delta(\cdot, \cdot)$ is the geodesic distance. It was proven in [37] that the Riemannian median defined by (4.1.3) exists and is unique in the case of a non-positively curved manifold such as $\mathcal{S}^{\mathrm{n}}_{++}$. The divergence-based median can be defined in a similar way when δ is the square root of divergence. The cost function in (4.1.3) is not differentiable at the data matrices, i.e., $X = A_i$ for $i = 1, \ldots, K$.

The computation of medians on $\mathcal{S}^{\mathrm{n}}_{++}$ has not received as much attention as the mean [21, 37, 84, 89]. Fletcher et al. [37] generalized the Weiszfeld-Ostresh's algorithm to the Riemannian median computation on an arbitrary manifold, and proved that the algorithm converges to the unique solution when it exists. Charfi et al. [21] considered the computation of medians based on the geodesic distance, Log-Euclidean distance and the Stein divergence. An Euclidean steepest descent method and a fixed point algorithm are proposed. However, the Euclidean steepest descent method is not appropriate since each iterate is not guaranteed to stay on $\mathcal{S}^{\mathrm{n}}_{++}$. Moreover, they did

not specify a stepsize selection rule for the steepest descent method. The Stein divergence median is also studied in [84], and they provided a convergence proof of the fixed point iteration in [21]. A median based on the total Kullback-Leibler divergence is proposed in [89], which has a closed form expression. We present a brief summary of previous work in Table 4.1.

Table 4.1: A summary of previous work for matrix medians using different distances/divergences.

Distance & Divergence	Reference	Closed-form expression/Algorithms
Affine-invariant distance	[21, 36, 37]	• Euclidean steepest descent • Riemannian Weiszfeld and Ostresh's algorithm
Log-Euclidean distance.	[21]	• Euclidean steepest descent • fixed point iteration
Stein divergence	[21, 84]	• Euclidean steepest descent • fixed point iteration
total Kullback-Leibler divergence	[89]	• closed-form expression

In this chapter, we apply the Riemannian optimization techniques on $\mathcal{S}_{++}^{\mathrm{n}}$ developed in Chapter 2 to handle the SPD median computation problem. Since the median cost function (4.1.3) is nonsmooth at the data matrices, we exploit recent developments in Riemannian optimization for nonsmooth functions to compute the median, including the modified Riemannian BFGS for nonsmooth functions in [46, Section 7] and a Riemannian version of nonsmooth BFGS in [45]. In addition, we modified and adapted LRBFGS for SPD median computation.

The main contributions of this dissertation for the SPD median computation are:

- The exploitation of smooth and nonsmooth versions of Riemannian quasi-Newton algorithms for SPD median computation.
- The systematic empirical investigation of the performance of proposed algorithms, and compare with the Riemannian version of Weiszfeld's method in [37].
- The investigation of the sided medians based on the log-determinant α-divergence.

4.2 Description of the L^1 Riemannian median computation methods

In this section, we present the algorithms for SPD median computation, including the Riemannian version of Weiszfeld's algorithm [37], a modified Riemannian BFGS for nonsmooth functions in [46, Section 7], a Riemannian version of nonsmooth BFGS in [45], and two adapted versions of limited-memory RBFGS for nonsmooth functions.

4.2.1 The Riemannian version of Weiszfeld's algorithm

In the Euclidean space, the geometric median of a set of points can be computed by an iterative algorithm introduced by Weiszfeld [92] and later improved by Ostresh [77]. Fletcher et al. [37] generalized the Weiszfeld procedure to arbitrary manifold, which is summarized in Algorithm 8. It was proven in [37] that Algorithm 8 converges to the Riemannian median if the stepsize α satisfies $0 < \alpha \leq 2$.

Algorithm 8 Riemannian Weiszfeld's Algorithm for the SPD Riemannian median computation

Input: $A_i = L_{A_i} L_{A_i}^T$; tolerance for stopping criteria ϵ; initial iterate $x_0 \in \mathcal{S}_{++}^{\mathrm{n}}$;

$k = 0$, $\alpha = 1$;
repeat
  for $i = 1, \ldots, K$ **do**
    Evaluate $\delta_R(x_k, A_i)$;
  end for
  if $x_k \neq A_1, \ldots, A_K$ **then**
    $\eta_k = -\sum_{i=1}^{K} \frac{1}{\delta(x_k, A_i)} \operatorname{Exp}_{x_k}^{-1}(A_i) \cdot (\sum_{i=1}^{K} \frac{1}{\delta(x_k, A_i)})^{-1}$;
  else if $x_k = A_j$ **then**
    $\eta_k = -\operatorname{Exp}_{x_k}^{-1}(A_j)$;
  end if
  Compute $x_{k+1} = \operatorname{Exp}_{x_k}(\alpha \eta_k)$;
  $k = k + 1$;
end(repeat)

We extend the Riemannian Weiszfeld's procedure to compute the median based on the LogDet α-divergence, which is described in Algorithm 9. However, the algorithm fails to converge in some numerical experiments if we use stepsize 1.

Algorithm 9 Riemannian Weiszfeld's algorithm for the SPD α-divergence median computation

Input: A_i; tolerance for stopping criteria ϵ; initial iterate $x_0 \in \mathcal{S}^{\mathrm{n}}_{++}$;

$k = 0$, $t = 1$;
repeat
  for $i = 1, \ldots, K$ **do**
    Evaluate $\delta_{\mathrm{LD},\alpha}(x_k, A_i)$;
  end for
  if $x_k \neq A_1, \ldots, A_K$ **then**
    $\eta_k = -\frac{1}{1-\alpha} \sum_{i=1}^{K} \frac{1}{\delta_{\mathrm{LD},\alpha}(x_k, A_i)} \{X(\frac{1-\alpha}{2} A_i + \frac{1+\alpha}{2} X)^{-1} X - X\} \cdot (\sum_{i=1}^{K} \frac{1}{\delta_{\mathrm{LD},\alpha}(x_k, A_i)})^{-1}$;
  else if $x_k = A_j$ **then**
    $\eta_k = -\frac{2}{1-\alpha} \{X(\frac{1-\alpha}{2} A_j + \frac{1+\alpha}{2} X)^{-1} X - X\}$;
  end if
  Compute $x_{k+1} = \mathrm{Exp}_{x_k}(t\eta_k)$;
  $k = k + 1$;
end(repeat)

4.2.2 A modified Riemannian BFGS method

For partly smooth functions, a stationary point can not be specified by the norm of the gradient. Clarke generalized the notion of gradient for partly smooth functions [28]. Recall that if $G : \mathbb{R}^n \to \mathbb{R}$ is a locally Lipschitz function, the Clarke generalized directional derivative of G at the point $x \in \mathbb{R}^n$ in the direction v, denoted by $G^\circ(x; v)$, is defined by

$$G^\circ(x; v) = \limsup_{y \to x, t \downarrow 0} \frac{G(y + tv) - G(y)}{t}.$$

The generalized subdifferential of G at x, denoted by $\partial G(x)$, is defined as the set

$$\partial G(x) := \{\xi \in X : \langle \xi, v \rangle \leq G^\circ(x; v) \text{ for all } v \in \mathbb{R}^n\}.$$

Clarke defines the generalized stationary point to be $x^\star$ where $G^\circ(x^\star; v) \geq 0$ for all $v \in \mathbb{R}^n$.

Let $f : \mathcal{M} \to \mathbb{R}$ be a locally Lipschitz function on a Riemannian manifold $\mathcal{M}$. For $x \in \mathcal{M}$, let $\hat{f}_x = f \circ R_x$ denote the restriction of the pullback $\hat{f} = f \circ R$ to $\mathrm{T}_x \mathcal{M}$. The Clarke generalized directional derivative of f at x in the direction $p \in \mathrm{T}_x \mathcal{M}$, denoted by $f^\circ(x; p)$, is defined by $f^\circ(x; p) = \hat{f}^\circ_x(0_x; p)$, where $\hat{f}^\circ_x(0_x; p)$ denotes the Clarke generalized directional derivative of $\hat{f}_x : \mathrm{T}_x \mathcal{M} \to \mathbb{R}$ at 0_x in the direction $p \in \mathrm{T}_x \mathcal{M}$. Therefore, the generalized subdifferential of f at x is

defined by $\partial f(x) = \partial \hat{f}_x(0_x)$. A point is a stationary point of f if $0 \in \partial f(x)$. A necessary condition that f achieves a local minimum at x is that x is a stationary point of f.

A Riemannian BFGS has been modified for nonsmooth functions on a manifold in [46, Section 7.3]. The two most important modifications are related to the line search algorithm and the stopping criterion. A nonsmooth Armijo condition is used to ensure a sufficient decrease in the objective function and a nonsmooth curvature condition is used to rule out unacceptably small stepsize.

Definition 4.2.1 (Armijo condition)**.** *Let $f : \mathcal{M} \to \mathbb{R}$ be a locally Lipschitz function on a Riemannian manifold $\mathcal{M}$ with a retraction R, $x \in \mathcal{M}$ and $p \in \mathrm{T}_x \mathcal{M}$. The stepsize α satisfies the Armijo condition if the following inequality holds for constant $c_1 \in (0, 1)$*

$$f(R_x(\alpha p)) - f(x) \leq c_1 \alpha f^\circ(x; p). \tag{4.2.1}$$

Definition 4.2.2 (Curvature condition)**.** *The stepsize α satisfies the curvature condition if the following inequality holds for constant $c_2 \in (c_1, 1)$,*

$$\sup_{\xi \in \partial f(R_x(\alpha p))} \langle \xi, \frac{1}{\beta_{\alpha p}} \mathcal{T}_{x \to R_x(\alpha p)}(p) \rangle \geq c_2 f^\circ(x; p), \tag{4.2.2}$$

where c_1 is the Armijo constant, $\beta_{\alpha p} = \|\alpha p\| / \|\mathcal{T}_{R_{\alpha p}}(\alpha p)\|$.

Lemma 4.2.1. *Let $f : \mathcal{M} \to \mathbb{R}$ be a locally Lipschitz function on a Riemannian manifold $\mathcal{M}$ and the function W defined by*

$$W(\alpha) := f(R_x(\alpha p)) - f(x) - c_2 \alpha f^\circ(x; p), \tag{4.2.3}$$

where $c_2 \in (c_1, 1)$, $x \in \mathcal{M}$ and $p \in \mathrm{T}_x \mathcal{M}$, be increasing on a neighborhood of some α_0, then α_0 satisfies the curvature condition.

Definition 4.2.3 (Wolfe condition)**.** *Let $f : \mathcal{M} \to \mathbb{R}$ be a locally Lipschitz function and $p \in \mathrm{T}_x \mathcal{M}$. We say α satisfies the Wolfe condition if it satisfies the Armijo and curvature conditions.*

A line search algorithm satisfying the Wolfe condition for partly smooth functions is proposed in [45] and is summarized in Algorithm 10. A modified RBFGS based on the line search 10 is given in Algorithm 12.

The modified RBFGS makes an assumption that the cost function is differentiable at all the iterates and thus the search direction is the same as the RBFGS for smooth cost functions. However,

for partly smooth functions, we cannot expect the norm of gradients go to zero. A modification to the stopping criterion is as follows.

Let J be a positive integer which is larger than the dimension of the manifold. Define $j_0 = 1$, $G_0 = \{\text{grad}\, f_0\}$ and, for $k = 1, 2, \ldots$ define

- $j_k = 1$, $G_k = \{\text{grad}\, f_k\}$ if $\|R_{x_{k-1}^{-1}}(x_k)\| > \epsilon$,
- $j_k = j_{k+1} + 1$, $G_k = \{\text{grad}\, f^{(k)}_{k-j_k+1}, \ldots, \text{grad}\, f^{(k)}_{k-1}, \text{grad}\, f^{(k)}_k\}$ if $\|R_{x_{k-1}^{-1}}(x_k)\| \leq \epsilon$, and $j_{k-1} < J$,
- $j_k = J$, $G_k = \{\text{grad}\, f^{(k)}_{k-J+1}, \ldots, \text{grad}\, f^{(k)}_{k-1}, \text{grad}\, f^{(k)}_k\}$ if $\|R_{x_{k-1}^{-1}}(x_k)\| \leq \epsilon$, and $j_{k-1} < J$,

where $\text{grad}\, f_i^{(j)} = \mathcal{T}_{R_{x_i}^{-1}(x_j)} \text{grad}\, f(x_i)$. G_k is a set of gradients evaluated at points near x_k. The modified RBFGS stops if the shortest length vector in the convex hull of G_k, i.e., $d_k = \arg\min\{\|d\| : d \in \text{conv}G_k\}$, is sufficiently small.

Algorithm 10 A line search algorithm for partly smooth function; $\alpha = Line(x, p, g, P, c_1, c_2)$

Input: $x \in \mathcal{M}$, a descent direction $p \in \mathrm{T}_x\mathcal{M}$ with $p = -Pg$, where $g \in \partial_\epsilon f(x)$ and P is a positive definite matrix and $c_1 \in (0, 1)$, $c_2 \in (0, 1)$, $a, b \in \mathbb{R}$.

Set $\alpha_0 = 0$, $\alpha_{\max} < l(\mathcal{M})$, $\alpha_1 = 1$, $i = 1$;
repeat
  Evaluate $A(\alpha_i) = f(R_x(\alpha_i p)) - f(x) - c_1\alpha_i\langle p, g\rangle$;
  if $A(\alpha_i) > 0$ **then**
    α must be obtained by $Zoom(x, p, g, P, \alpha_{i-1}, \alpha_i, c_1, c_2)$;
    Stop;
  end if
  Compute $\xi \in \partial f(R_x(\alpha_i p))$;
  if $\langle \xi, \frac{1}{\beta_{\alpha_i p}}\mathcal{T}_{x \to R_x(\alpha_i p)}(p)\rangle \geq c_2\langle p, g\rangle$ **then**
    $\alpha = \alpha_i$;
    Stop;
  else
    $\alpha_{i+1} = \min(2\alpha_i, \alpha_{\max})$;
  end if
  $i = i + 1$;
end(repeat)

Algorithm 11 $\alpha = Zoom(x, p, g, P, a, b, c_1, c_2)$

Input: $x \in \mathcal{M}$, a descent direction $p \in \mathrm{T}_x\mathcal{M}$ with $p = -Pg$, where $g \in \partial_\epsilon f(x)$ and P is a positive definite matrix and $c_1 \in (0,1)$, $c_2 \in (0,1)$, $a, b \in \mathbb{R}$.

$i = 1$, $a_1 = a$, $b_1 = b$;
repeat
  $\alpha_i = \frac{a_i + b_i}{2}$;
  Evaluate $A(\alpha_i) = f(R_x(\alpha_i p)) - f(x) - c_1\alpha_i\langle p, g\rangle$;
  if $A(\alpha_i) > 0$ **then**
    $a_{i+1} = a_i$, $b_{i+1} = \alpha_i$;
  else
    Compute $\xi \in \partial f(R_x(\alpha_i p))$;
    if $\langle \xi, \frac{1}{\beta_{\alpha_i p}}\mathcal{T}_{x \to R_x(\alpha_i p)}(p)\rangle \geq c_2\langle p, g\rangle$ **then**
      $\alpha = \alpha_i$;
      Stop;
    else
      $a_{i+1} = \alpha_i$, $b_{i+1} = b_i$;
    end if
  end if
  $i = i + 1$;
end(repeat)

4.2.3 A modified limited-memory Riemannian BFGS method

To adapt LRBFGS for partly smooth functions, we make similar modifications to smooth LRBFGS as RBFGS in Section 4.2.2. A modified LRBFGS is given in Algorithm 13.

4.2.4 A nonsmooth Riemannian BFGS method

Recently, a version of nonsmooth Riemannian BFGS method is proposed in [45, Algorithm 8]. The main difference between the modified RBFGS in Algorithm 12 and the nonsmooth RBFGS is the search direction, see [45] for details. We present the nonsmooth RBFGS in Algorithm 15.

4.2.5 A nonsmooth limited-memory Riemannian BFGS method

We also consider a nonsmooth version of limited-memory RBFGS, which is given in Algorithm 16. As we know, LRBFGS is appropriate for large-size problems. However, the nonsmooth

[0]If the locking condition is imposed, then $y_k^{(k+1)} = \operatorname{grad} f(x_{k+1})/\beta_k - \mathcal{T}_{\alpha_k\eta_k}\operatorname{grad} f(x_k)$, where $\beta_k = \|\alpha_k\eta_k\|/\|\mathcal{T}_{R_{\alpha_k\eta_k}\alpha_k\eta_k}\|$.

Algorithm 12 A modified RBFGS algorithm [46, Section 7.3]

Input: A starting point $x_1 \in \mathcal{M}$, $P_1 = I$, Wolfe condition constants $c_1 \in (0,1)$, $c_2 \in (c_1, 1)$;

1: $k = 1$;

2: Compute $g_k \in \partial f(x_k)$;

3: **repeat**

4: $p_k = -P_k g_k$;

5: Set $x_{k+1} = R_{x_k}(\alpha_k p_k)$ where $\alpha_k > 0$ is computed from the line search procedure to satisfy the Wolfe conditions using Algorithm 10

$$\alpha_k = Line(x_k, p_k, g_k, P_k, c_1, c_2);$$

6: Set $x_{k+1} = R_{x_k}(\alpha_k p_k)$;

7: Compute $g_{k+1} \in \partial f(x_{k+1})$;

8: Define $s_k := \mathcal{T}_{S_{\alpha_k p_k}}(\alpha_k p_k)$, $y_k := \frac{1}{\beta_{\alpha_k p_k}} g_{k+1} - \mathcal{T}_{S_{\alpha_k p_k}}(g_k)$, $s_k := s_k + \max(0, \frac{1}{\Lambda} - \frac{s_k^\flat y_k}{y_k^\flat y_k})$;

9: Define the linear operator $P_{k+1} : \mathrm{T}_{x_{k+1}} \mathcal{M} \to \mathrm{T}_{x_{k+!}} \mathcal{M}$ by

$$P_{k+1} = \mathcal{V}_k^\flat \tilde{P}_k \mathcal{V}_k + \rho_k s_k s_k^\flat, \tag{4.2.4}$$

where $\rho_k = 1/g(y_k, s_k)$, $\mathcal{V}_k = \mathrm{id} - \rho_k y_k s_k^\flat$ and $\tilde{P}_k = \mathcal{T}_{S_{\alpha_k p_k}} \circ P_k \circ \mathcal{T}^{-1}_{S_{\alpha_k p_k}}$;

10: $k = k + 1$;

11: **end(repeat)**

LRBFGS loses some of its appeal, since it requires solving a convex hull problem.

4.3 Numerical experiments

In this section, we compare the performances of different algorithms on various data sets, including the Riemannian Endre Weiszfeld's algorithm (EW) in Algorithm 8, the smooth RBFGS combined with the Wolfe line search (RBFGS-Wolfe), the smooth LRBFGS combined with the Wolfe line search (LRBFGS-Wolfe), the modified RBFGS for partly smooth functions (RBFGS-WolfeLP) in Algorithm 12, the modified LRBFGS for partly smooth functions (LRBFGS-WolfeLP) in Algorithm 13, the nonsmooth RBFGS in Algorithm 15 (NS-RBFGS), and the nonsmooth LRBFGS in Algorithm 16 (NS-LRBFGS).

For simplicity of notation, throughout this section we denote the number, dimension, and condition number of the matrices by K, n, and κ respectively. Regarding the parameter setting, we set Wolfe parameter $c_1 = 10^{-4}$ and $c_2 = 0.999$. For the modified RBFGS and LRBFGS, ϵ and

J are set to be 10^{-6} and $2d$, where $d = n(n+1)/2$ is the dimension of $\mathcal{S}^{\mathrm{n}}_{++}$. The parameters for the nonsmooth RBFGS and LRBFGS are set as follows: $\epsilon_1 = 10^{-2}$, $\delta_1 = 10^{-4}$, $\theta_\epsilon = 10^{-2}$, $\theta_\delta = 10^{-2}$, $\lambda = 10^{-2}$, $\Lambda = 10^2$. For three versions of LRBFGS, we take $m = 2$. Unless otherwise specified, our choice of the initial iterate is the arithmetic-harmonic mean of data matrices. We run the algorithms until they reach their highest accuracy. This allows the algorithms to reach the noise floor after enough iterations.

All experiments are performed on the Florida State University HPC system with 24 Intel(R) Xeon(R) CPU E5-2680 v3 processor 2.5GHz. All the experiments are carried out using C++, compiled with gcc-4.7.x.

4.3.1 Comparison of performances between different algorithms for SPD Riemannian median computation

As a first test, we investigate the performance of all 7 algorithms for Riemannian median computation on datasets with different distributions. We will observe that the performance algorithms is influenced by the distributions of datasets. For each setting of parameters, 50 random runs with the same seeds are used. The test datasets $\{A_1, \ldots, A_K\}$ are constructed as follows.

I. Generate 100 data matrices that belong to a small ball $B(I, r)$ of radius r centered at the identity matrix I. More specifically, $A_i's$ are generated as follows:

   a. Generate symmetric matrices $\eta_i \in \mathrm{T}_I(\mathcal{S}^{\mathrm{n}}_{++})$;

   b. Normalize $\eta_i = \eta_i/\|\eta_i\|$;

   c. $A_i = \mathrm{Exp}_O(t\eta_i)$, $t \in (0, 1)$.

II. Generate 100 data matrices that belong to a small ball $B(O, r)$, where O is a random ill-conditioned matrix with condition number around 10^5.

III. Generate 95 well-conditioned random matrices with condition number less than 10, and add 5 ill-conditioned random matrices with condition number around 10^5 as outliers.

IV. Generate 95 ill-conditioned random matrices with condition number around 10^5, and add 5 well-conditioned random matrices with condition number around 1 as outliers.

V. Generate 100 random matrices that are separated into 4 clusters, and each cluster contains 25 matrices. More specifically, $A_i's$ are generated as follows:

   a. Generate 4 random matrices O_c that are away from each other, and the between-cluster distance is between 5 and 100.

b. For each O_c, generate 25 random matrices that belong to $B(O_c, r)$, and the within-cluster distance is between 10^{-2} and 10^{-3}.

Figure 4.3-4.5 display the performance results of different algorithms running on small dimension matrices with $K = 100$ and $n = 3$. Figure 4.3 reports the results tested on data matrices that belong to a ball $B(O, r)$ with radius $r = 0.1$. That is, $\delta(A_i, O) \leq 0.1$ for $i = 1, \ldots, K$. The center O is the identity matrix for the two plots on the top row and O is an ill-conditioned matrix with $\kappa = 10^5$ for the two plots on the bottom row. We observe that the smooth RBFGS and modified RBFGS perform similarly, as well as LRBFGS. Even though the nonsmooth RBFGS requires similar number of iterations as RBFGS and modified RBFGS, it requires much more time. One major time consuming factor is the requirement of solving a convex hull problem. The nonsmooth LRBFGS performs poorly in terms of both number of iterations and computation time. The performance the Riemannian Weiszfeld's algorithm is outperformed by the smooth and modified quasi-Newton algorithms, but better than the nonsmooth versions.

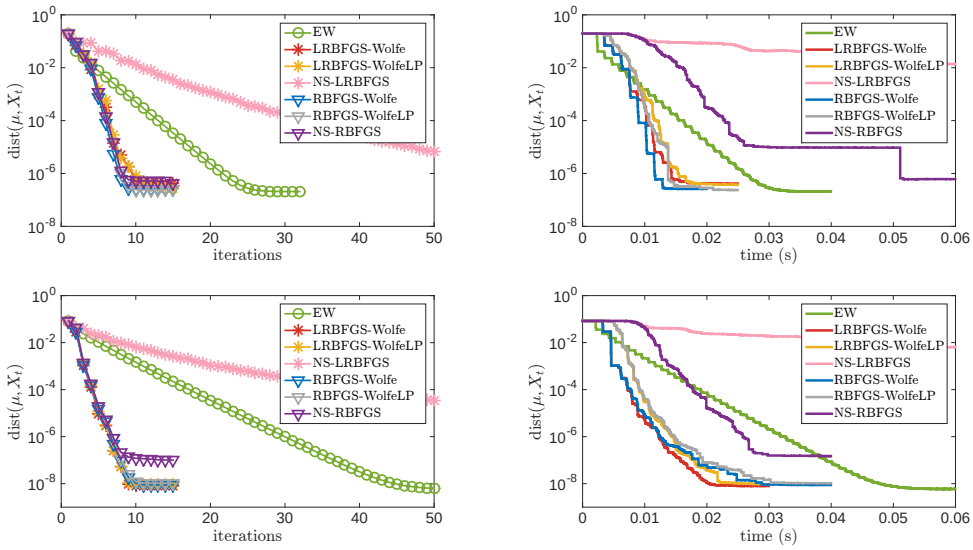


Figure 4.3: Evolution of averaged distance between current iterate and the exact Riemannian median with respect to time and iterations for $K = 100$ and $n = 3$. Top row: A_i's belong to a small ball $B(I, r)$ centered at the identity matrix; Bottom row: A_i's belong to a small ball centered at an ill-conditioned matrix.

Figure 4.4 reports the results tested on datasets in presence of outliers. For the two plots on the top row, the dataset contains 95 well-conditioned matrices with $1 \leq \kappa(A_i) \leq 2$ and 5 ill-conditioned

outliers with $10^4 \leq \kappa \leq 10^6$. For the two plots on the bottom row, the dataset contains 95 ill-conditioned matrices with $10^4 \leq \kappa(A_i) \leq 10^6$ and 5 well-conditioned outliers with $1 \leq \kappa(A_i) \leq 2$. In both cases, the nonsmooth RBFGS and LRBFGS are outperformed by the other 5 algorithms in terms of computation time. It is also observed that in the presence of outliers, the smooth LRBFGS and modified LRBFGS are preferred than RBFGS in terms of both number of iterations and computation time. Even though the size of matrices is small ($n = 3$), it turns out that going for limited-memory version offers a computational advantage.

Figure 4.5 reports the results tested on dataset in which matrices are clustered into 4 groups. Each cluster contains 25 matrices. It is observed that the Riemannian Weiszfeld's algorithm requires a significantly large number of iterations to converge. The modified LRBFGS is clearly the winner in terms of number of iterations and computation time.

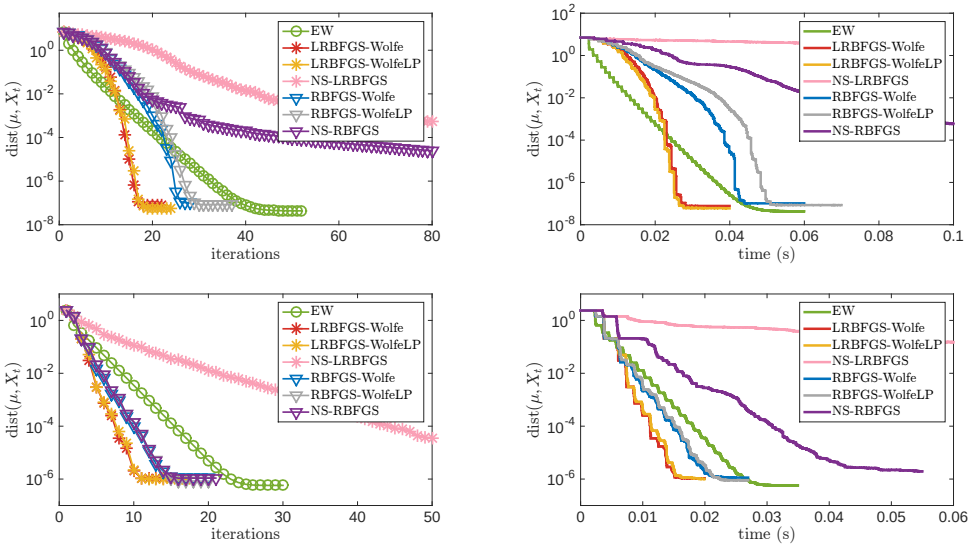


Figure 4.4: Evolution of averaged distance between current iterate and the exact Riemannian median with respect to time and iterations for $K = 100$ and $n = 3$. Top row: well conditioned A_i's with 5% ill-conditioned outliers; Bottom row: ill conditioned A_i's with 5% well-conditioned outliers.

Figure 4.6-4.8 display the performance of results of different algorithms running on large size matrices, i.e., taking ($K = 100$, $n = 100$). We did not compare two nonsmooth quasi-Newton algorithms, since solving a convex hull problem in high dimension space is very memory consuming

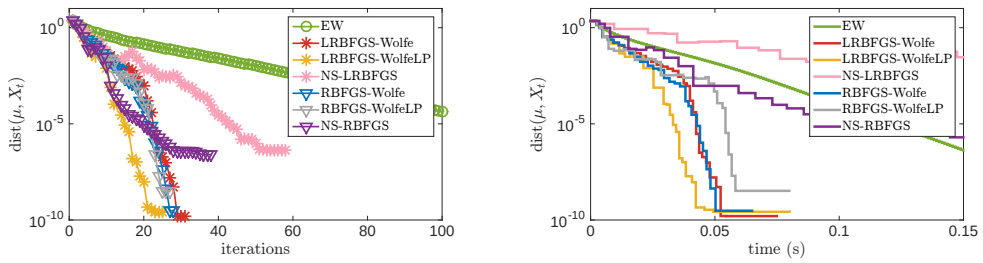


Figure 4.5: Evolution of averaged distance between current iterate and the exact Riemannian median with respect to time and iterations for $K = 100$ and $n = 3$. A_i's are separated into 4 clusters.

and time consuming. Note that when $n = 100$, the dimension of $\mathcal{S}^{\mathrm{n}}_{++}$ is $d = n(n+1)/2 = 5050$. We also test the algorithms on 5 different distributed datasets as the case of $n = 3$.

When all the data matrices belong to a ball, it is observed from Figure 4.6 that the initial iterate is very close to the true solution. 4 Riemannian optimization algorithms reach their noise floors in just a few steps, while the Riemannian Weiszfeld's algorithm requires much larger number of iterations. In terms of computation time, the RBFGS and its modified version requires much more time than that of limited-memory versions. Among two versions of LRBFGS, the smooth one requires the least computation time.

Figure 4.7 displays the results from datasets containing outliers. The convergence behavior of the algorithms is very similar as that displayed in Figure 4.4 where $n = 3$. However, the computation time of the smooth RBFGS and modified RBFGS increases dramatically. In Figure 4.8, the data matrices are clustered into 4 groups. It is observed that the Riemannian Weiszfeld's algorithm requires a large number of iterations to converge. Even though RBFGS requires similar number of iterations as LRBFGS, RBFGS takes much more time. It is shown empirically that LRBFGS is appropriate for large-size problems.

4.3.2 Comparison of performances between different algorithms for SPD LogDet α-divergence median computation

As a second test, we investigate the performance of all 7 algorithms for SPD LogDet α-divergence median computation. Different values of α are considered. The experiments are designed in the same way as Section 4.3.1. We use the same datasets as in Section 4.3.1.

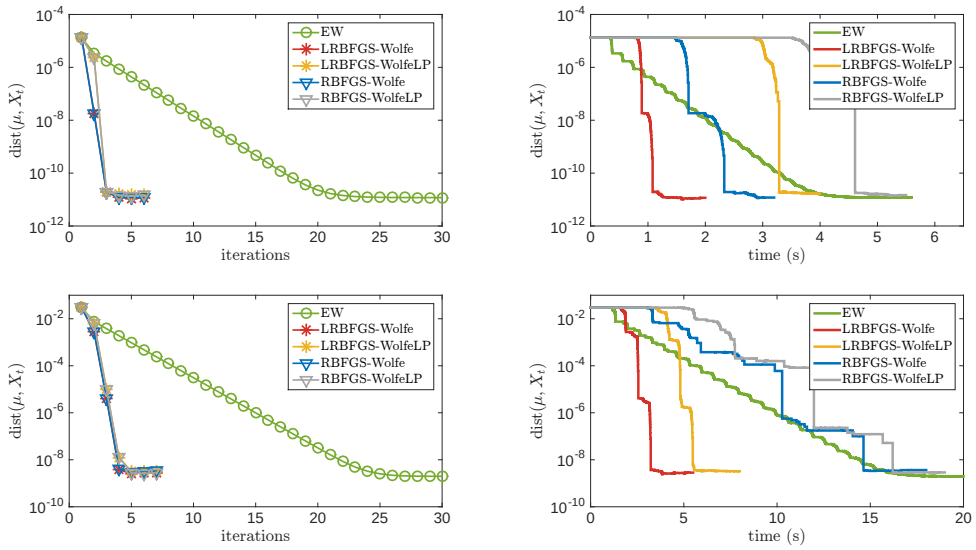


Figure 4.6: Evolution of averaged distance between current iterate and the exact Riemannian median with respect to time and iterations for $K = 100$ and $n = 100$. Top row: A_i's belong to a small ball $B(I, r)$ centered at the identity matrix; Bottom row: A_i's belong to a small ball centered at an ill-conditioned matrix.

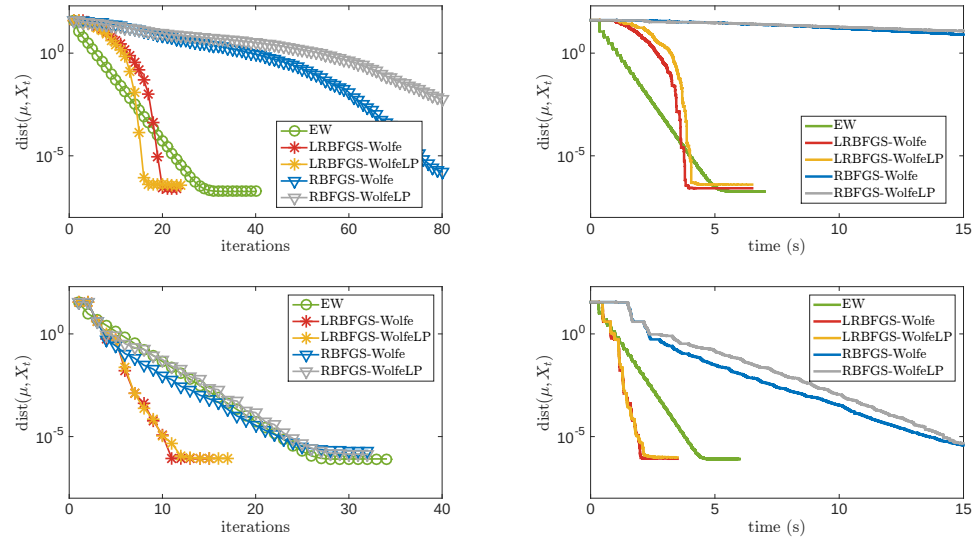


Figure 4.7: Evolution of averaged distance between current iterate and the exact Riemannian median with respect to time and iterations for $K = 100$ and $n = 100$. Top row: well conditioned A_i's with 5% ill-conditioned outliers; Bottom row: ill conditioned A_i's with 5% well-conditioned outliers.

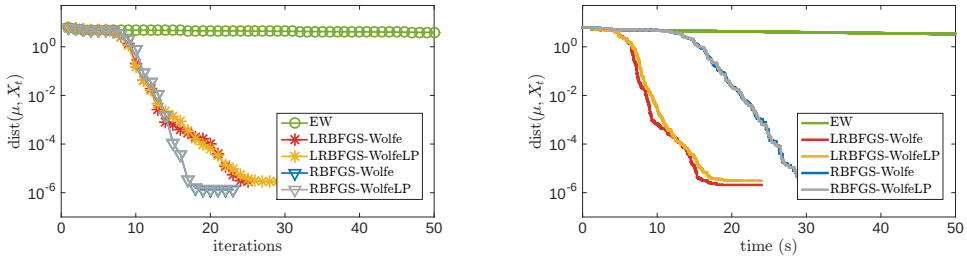


Figure 4.8: Evolution of averaged distance between current iterate and the exact Riemannian median with respect to time and iterations for $K = 100$ and $n = 100$. A_i's are separated into 4 clusters.

Figure 4.9-4.11 report the results of tests conducted on datasets with $K = 100$, $n = 3$ and $\alpha = 0$. For Riemannian quasi-Newton algorithms, the numerical results are similar with what we observed in the computation of Riemannian median in previous section. However, it is observed that the Riemannian Weiszfeld's procedure fails to converge in a large number of numerical experiments.

When the data matrices belong to a ball as shown in Figure 4.9, we observe that the smooth RBFGS and modified RBFGS perform similarly, which is also the case for LRBFGS. Even though the nonsmooth RBFGS requires similar number of iterations as RBFGS and modified RBFGS, it requires much more time. The nonsmooth LRBFGS performs poorly in terms of both number of iterations and computation time. When the data matrices are centered at the identity matrix, those 4 algorithms are competitive. When the data matrices are centered at an ill-conditioned matrix, we observe a slight advantage for the smooth RBFGS and the modified RBFGS as they reach a higher accuracy than LRBFGS. In the presence of outliers as shown in Figure 4.10, the smooth LRBFGS and the modified LRBFGS win over RBFGS in terms of number of iterations and computation time. When the data matrices are clustered into 4 clusters, the smooth LRBFGS and the modified LRBFGS are still the winner.

More experiment results for $\alpha = -0.5$ and $\alpha = 0.5$ are given in Figure 4.12 and Figure 4.12.

4.3.3 Comparison between Riemannian means and medians

As the last experiment, we use tensor data, i.e., 3×3 SPD matrices to illustrate the robustness of the Riemannian median. A tensor can be visualized as an ellipsoid, whose axes point along the eigenvectors and the lengths of the axes are given by the corresponding eigenvalues. For tensor

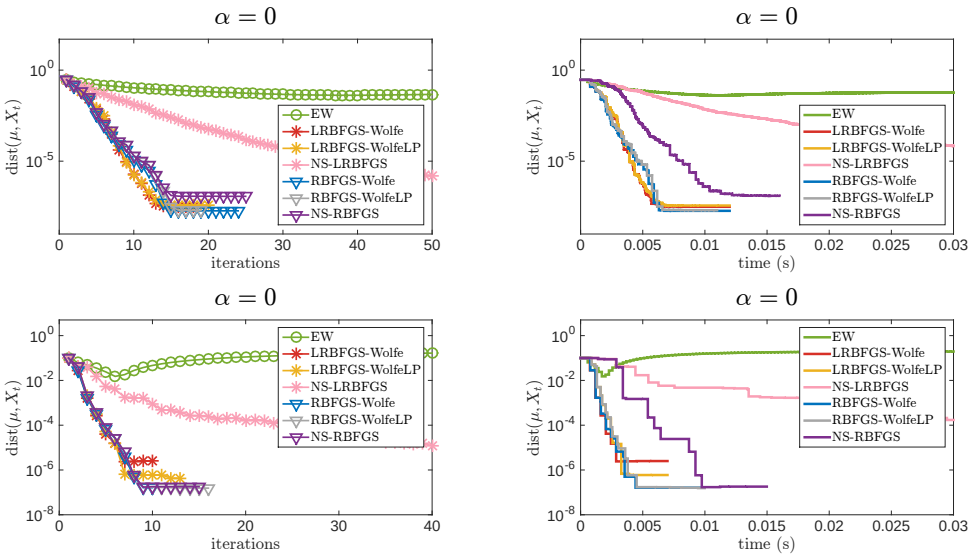


Figure 4.9: Evolution of averaged distance between current iterate and the exact LogDet α-divergence median with respect to time and iterations for $K = 100$ and $n = 3$. Top row: A_i's belong to a small ball $B(I, r)$ centered at the identity matrix; Bottom row: A_i's belong to a small ball centered at an ill-conditioned matrix.

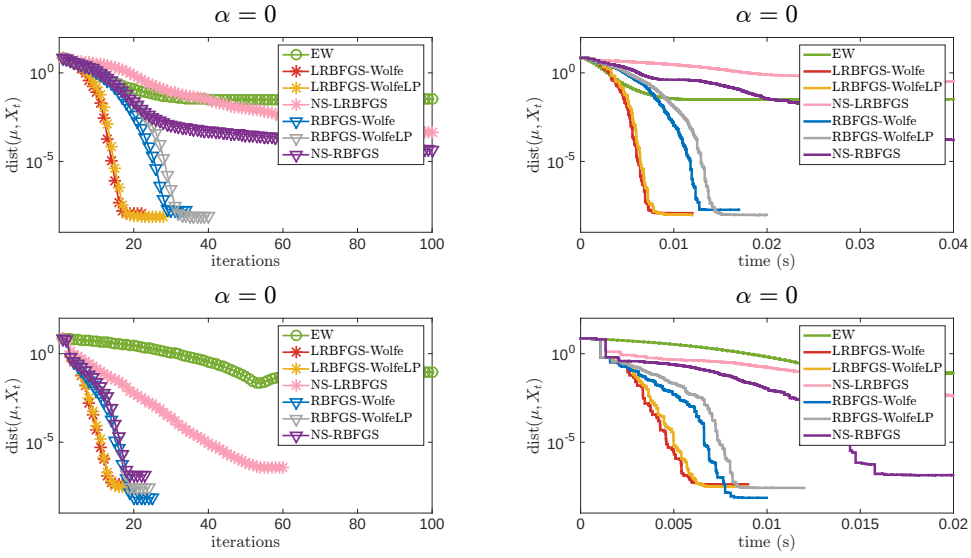


Figure 4.10: Evolution of averaged distance between current iterate and the exact LogDet α-divergence median with respect to time and iterations for $K = 100$ and $n = 3$. Top row: well conditioned A_i's with 5% ill-conditioned outliers; Bottom row: ill conditioned A_i's with 5% well-conditioned outliers.

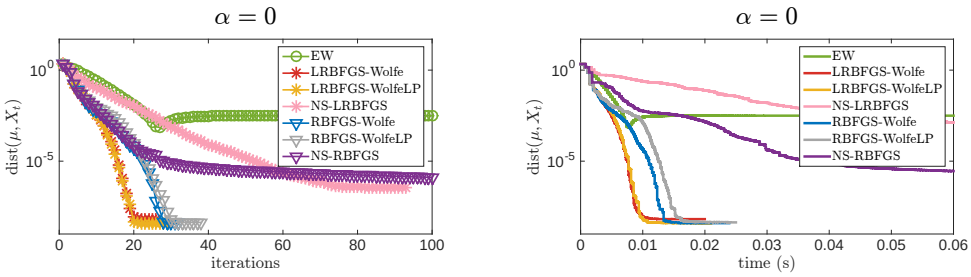


Figure 4.11: Evolution of averaged distance between current iterate and the exact LogDet α-divergence median with respect to time and iterations for $K = 100$ and $n = 3$. A_i's are separated into 4 clusters.

visualization, we resort to the fanDTasia ToolBox[1] developed by [14]. The tensors are colored based on the direction of the major eigenvector, i.e., eigenvector associated with the largest eigenvalue. The color brightness is a measure of tensor anisotropy. That is, more brightness implies more anisotropy.

For each test, we generate 50 random 3×3 SPD matrices as following:

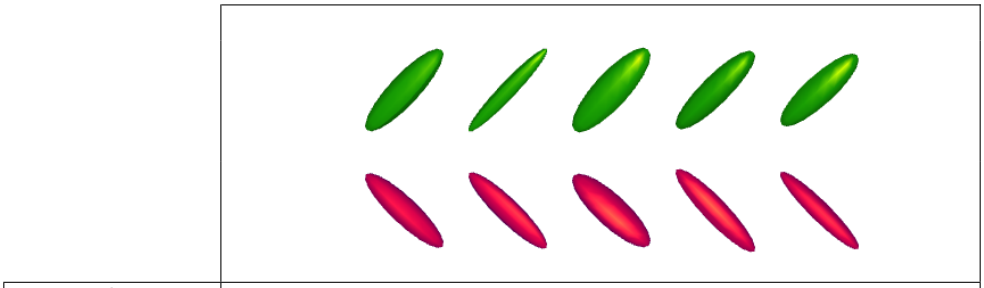
- Generate random orthonormal matrix O, whose columns are eigenvectors for A_i. Note that we use the same eigenvectors O for each A_i.
- Generate eigenvalues for each A_i with $D_i = (\lambda_1, \lambda_2, \lambda_3) \sim \mathcal{N}(\bar{\lambda}, \sigma^2 I)$.
- Compute $A_i = O D_i O^T$.
- Add Gaussian noise to the Cholesky factor of $A_i = L_i L_i^T$, i.e., $\tilde{L}_i = L_i + E_i$
- Compute $A_i = \tilde{L}_i \tilde{L}_i^T$.

Numerical experiments I. For the original data tensors, we take mean $\bar{\lambda} = (4, 1, 1)$ and standard deviation $\sigma = 0.2$. We generate outliers whose eigenvalues follow the same distribution as the original data but the major eigenvector is perpendicular to that of the original tensors. The original dataset contains 50 tensors. We compute the Riemannian medians and means of the tensor dataset with 0, 1, 5, 10, 25, and 50 outliers. (When 50 outliers are added to the dataset, they are not outliers anymore.) The results are displayed in Table 4.2. Shown in the top row of Table 4.2 are 5 samples of the original dataset (green) and 5 samples of the outliers (red). The resulting means and medians are colored in yellow.

[1] https://www.mathworks.com/matlabcentral/fileexchange/26997-fandtasia-toolbox

We can observe that the median is very robust to outliers. It preserves the shape of original tensors. As we are adding more and more outliers, such as 25, the shape of the median still preserves the shape of the major tensor group. The shape of the mean has been influenced.

Table 4.2: Comparison of the Riemannian medians and means for 3×3 SPD matrices based on the geodisic distance δ_R and the log-determinant α-divergence $\delta_{\mathrm{LD},\alpha}$ with $\alpha = 0$ and $\alpha = 0.5$. Shown in the top row are 5 samples of the original dataset (green) and 5 samples of the outliers (red). The major eigenvectors of the original tensors and the outliers are perpendicular to each other. The resulting means and medians are colored in yellow.

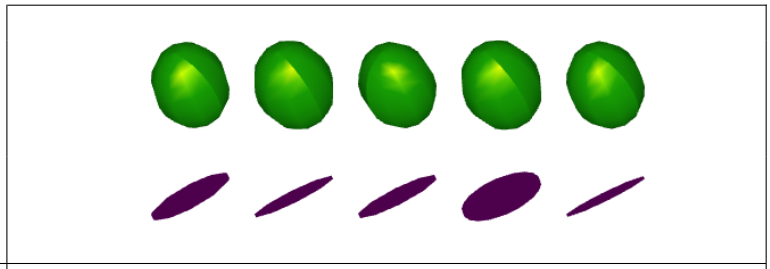


outliers		0	1	5	10	25	50
δ_R	Median						
	Mean						
$\delta_{\mathrm{LD},0}$	Median						
	Mean						
$\delta_{\mathrm{LD},0.5}$	Median						
	Mean						

Numerical experiments II. In the second experiment, we generate tensors in a similar way. We generate 50 well-conditioned tensors with mean $\bar{\lambda} = (5, 4, 4)$ and standard deviation

$\sigma = 0.2$. Then we generate ill-conditioned outliers with mean $\bar{\lambda} = (20, 1, 10^5)$ and the same standard deviation. The results are displayed in Table 4.3. Notice that the color of the outliers in Table 4.3 become very dark since they are ill-conditioned. As we are adding more and more ill-conditioned outliers, the median stays well conditioned.

Table 4.3: Comparison of the Riemannian medians and means for 3×3 SPD matrices based on the geodisic distance δ_R, and the log-determinant α-divergence $\delta_{\mathrm{LD},\alpha}$ with $\alpha = 0$ and $\alpha = 0.5$. Shown in the top row are 5 samples of the original dataset (green) and 5 samples of the outliers (red). The original tensors are well-conditioned with condition number ≤ 2, while the outliers are ill-conditioned with condition number $\approx 10^5$. The resulting means and medians are colored in yellow.



outliers		0	1	5	10	25	50
δ_R	Median						
	Mean						
$\delta_{ld,0}$	Median						
	Mean						
$\delta_{ld,0.5}$	Median						
	Mean						

4.4 Conclusions

In this chapter, we consider computing the median of a collection of SPD matrices based on the Riemannian geodesic distance and the log-determinant α-divergence. We exploit 3 versions of Riemannian quasi-Newton algorithms to handle this computational task, including the smooth version, the modified version and the nonsmooth version. We empirically and systematically investigate the performance of proposed Riemannian optimization algorithms and compare with the state-of-the-art Riemannian Weiszfelds method. We examine the performance results of compared algorithms on various datasets with different distributions and various choices of (K, n, κ). For the SPD Riemannian median computation, the smooth version of LRBFGS appears to be the method of choice in all cases. For the SPD LogDet α-divergence-based median computation, the modified LRBFGS performs the best.

Algorithm 13 A modified LRBFGS algorithm

Input: A starting point $x_1 \in \mathcal{M}$, $P_1 = I$, Wolfe condition constants $c_1 \in (0,1)$, $c_2 \in (c_1,1)$, an integer $m > 0$;

1: $k = 0$, $\gamma_0 = 1$, $l = 0$;
2: Compute $g_k \in \partial f(x_k)$;
3: **repeat**
4: $\mathcal{H}_k^0 = \gamma_k \,\mathrm{id}$. Obtain $p_k \in \mathrm{T}_{x_k}\mathcal{M}$ by the following algorithm, Step 5 to Step 13:
5: $q \leftarrow g_k$;
6: **for** $i = k-1, k-2, \ldots, k-l$ **do**
7: $\xi_i \leftarrow \rho_i g(s_i^{(k)}, q)$; $q \leftarrow q - \xi_i y_i^{(k)}$;
8: **end for**
9: $r \leftarrow \mathcal{H}_k^0 q$;
10: **for** $i = k-l, k-l+1, \ldots, k-1$ **do**
11: $\omega \leftarrow \rho_i g(y_i^{(k)}, r)$;
12: $r \leftarrow r + s_i^{(k)}(\xi_i - \omega)$;
13: **end for**
14: Set $p_k = -r$;
15: Set $x_{k+1} = R_{x_k}(\alpha_k p_k)$ where $\alpha_k > 0$ is computed from the line search procedure to satisfy the Wolfe conditions using Algorithm 10

$$\alpha_k = Line(x_k, p_k, g_k, P_k, c_1, c_2);$$

16: Set $x_{k+1} = R_{x_k}(\alpha_k p_k)$;
17: Compute $g_{k+1} \in \partial f(x_{k+1})$;
18: Define $s_k^{(k+1)} = \mathcal{T}_{\alpha_k\eta_k}\alpha_k\eta_k$, $y_k^{(k+1)} = \frac{1}{\beta_{\alpha_k p_k}} g_{k+1} - \mathcal{T}_{\alpha_k\eta_k} g_k$, $s_k := s_k + \max(0, \frac{1}{\Lambda} - \frac{s_k^\flat y_k}{y_k^\flat y_k})$;
19: Compute $a = g(y_k^{(k+1)}, s_k^{(k+1)})$ and $b = \|s_k^{(k+1)}\|^2$;
20: **if** $\frac{a}{b} \geq \lambda$ **then**
21: Compute $c = \|y_k^{(k+1)}\|^2$ and define $\rho_k = 1/a$ and $\gamma_{k+1} = a/c$;
22: Add $s_k^{(k+1)}$, $y_k^{(k+1)}$ and ρ_k into storage and if $l \geq m$, then discard vector pair $\{s_{k-l}^{(k)}, y_{k-l}^{(k)}\}$ and scalar ρ_{k-l} from storage, else $l \leftarrow l+1$; Transport $s_{k-l+1}^{(k)}, s_{k-l+2}^{(k)}, \ldots, s_{k-1}^{(k)}$ and $y_{k-l+1}^{(k)}, y_{k-l+2}^{(k)}, \ldots, y_{k-1}^{(k)}$ from $\mathrm{T}_{x_k}\mathcal{M}$ to $\mathrm{T}_{x_{k+1}}\mathcal{M}$ by $\mathcal{T}$, then get $s_{k-l+1}^{(k+1)}, s_{k-l+2}^{(k+1)}, \ldots, s_{k-1}^{(k+1)}$ and $y_{k-l+1}^{(k+1)}, y_{k-l+2}^{(k+1)}, \ldots, y_{k-1}^{(k+1)}$;
23: **else**
24: Set $\gamma_{k+1} \leftarrow \gamma_k$, $\{\rho_k, \ldots, \rho_{k-l+1}\} \leftarrow \{\rho_{k-1}, \ldots, \rho_{k-l}\}$, $\{s_k^{(k+1)}, \ldots, s_{k-l+1}^{(k+1)}\} \leftarrow \{\mathcal{T}_{\alpha_k\eta_k} s_{k-1}^{(k)}, \ldots, \mathcal{T}_{\alpha_k\eta_k} s_{k-l}^{(k)}\}$ and $\{y_k^{(k+1)}, \ldots, y_{k-l+1}^{(k+1)}\} \leftarrow \{\mathcal{T}_{\alpha_k\eta_k} y_{k-1}^{(k)}, \ldots, \mathcal{T}_{\alpha_k\eta_k} y_{k-l}^{(k)}\}$;
25: **end if**
26: $k = k+1$;
27: **end(repeat)**

Algorithm 14 A descent direction algorithm [45]; $(g_k, p_k) = Descent(x, \delta, c, \epsilon, P)$.

Input: $x \in \mathcal{M}$, $\delta > 0$, $c \in (0,1)$, $0 < \epsilon < l(\mathcal{M})$ and a positive definite matrix P.

Select arbitrary $v \in \partial_\epsilon f(x)$;
Set $W_1 = \{v\}$, $k = 1$;
Step 1: (Compute a descent direction)
Solve the following minimization problem

$$g_k = \arg\min_{v \in \text{conv}W_k} \langle Pv, v\rangle;$$

if $\|g_k\|^2 \leq \delta$ **then** Stop;
else
    Let $p_k = -Pg_k$;
end if
Step 2: (Stopping condition)
if $f(R_x(\frac{\epsilon p_k}{\|p_k\|})) - f(x) \leq \frac{-c\epsilon\langle Pg_k, g_k\rangle}{\|p_k\|}$ **then** Stop;
end if
Step 3: (Find a vector $v_{k+1} \in \partial_\epsilon f(x)$, which can be added to W_k)
$t \leftarrow \frac{b}{\|p_k\|}$, $\epsilon \leftarrow \frac{\epsilon}{\|p_k\|}$, $a \leftarrow 0$;
while $\langle v, \frac{1}{\beta_{tp_k}}\mathcal{T}_{x \to R_x(tp_k)}(p_k)\rangle + c\langle Pg_k, g_k\rangle \geq 0$ **do**
    select $v \in \partial f(R_x(tp_k))$;
    if $\langle v, \frac{1}{\beta_{tp_k}}\mathcal{T}_{x \to R_x(tp_k)}(p_k)\rangle + c\langle Pg_k, g_k\rangle < 0$ **then**
        $t = \frac{a+b}{2}$;
        Evaluate $h(t) = f(R_x(tp_k)) - f(x) + ct\langle Pg_k, g_k\rangle$, $h(b) = f(R_x(bp_k)) - f(x) + cb\langle Pg_k, g_k\rangle$;
        if $h(b) > h(t)$ **then**
            $a = t$
        else
            $b = t$
        end if
    end if
end while
$v_{k+1} = \beta_{tp_k}^{-1}\mathcal{T}_{x \leftarrow R_x(tp_k)}(v)$, $W_{k+1} = W_k \cup \{v_{k+1}\}$, $k = k+1$. Go to Step 1.

Algorithm 15 A nonsmooth Riemannian BFGS algorithm [45]

Input: A starting point $x_1 \in \mathcal{M}$, $c_1 \in (0,1)$, $c_2 \in (c_1,1)$, $\theta_\epsilon, \theta_\delta \in (0,1)$, $\delta_1 > 0$, $\epsilon_1 \in (0, l(\mathcal{M}))$, $k = 1$, $P_1 = I$, a bound $1/\Lambda > 0$ on $\frac{y_k^\flat s_k}{y_k^\flat y_k}$ and λ on $\frac{s_k^\flat y_k}{s_k^\flat s_k}$

Step 1: (Set new parameters) $s = 1$, $x_k^s = x_k$ and $P_k^s = P_k$;

Step 2: (Descent direction) $(g_k^s, p_k^s) = Descent(x_k^s, \delta_k, c_1, \epsilon_k, P_k^s)$

if $\|g_k^s\| = 0$ **then** Stop;

end if

if $\|g_k^s\|^2 \leq \delta_k$ **then**

set $\epsilon_{k+1} = \epsilon_k \delta_\epsilon$, $\delta_{k+1} = \delta_k \theta_\delta$, $x_{k+1} = x_k^s$, $P_{k+1} = P_k^s$, $k = k+1$. Go to Step 1;

else

$$\alpha = Line(x_k^s, p_k^s, g_k^s, P_k^s, c_1, c_2)$$

and construct the next iterate $x_k^{s+1} = R_{x_k^s}(\alpha p_k^s)$, $\xi_k \in \partial f(x_k^{s+1})$ and define $s_k := \mathcal{T}_{x_k^s \to R_{x_k^s}(\alpha p_k^s)}(\alpha p_k^s)$, $y_k := \frac{1}{\beta_{\alpha p_k^s}} \xi_k - \mathcal{T}_{x_k^s \to R_{x_k^s}(\alpha p_k^s)}(g_k)$, $s_k := s_k + \max(0, \frac{1}{\Lambda} - \frac{s_k^\flat y_k}{y_k^\flat y_k})$;

if $\frac{s_k^\flat y_k}{s_k^\flat s_k} \geq \lambda$ **then** Update

$$P_k^{s+1} = \mathcal{V}_k^\flat \tilde{P}_k^s \mathcal{V}_k + \rho_k s_k s_k^\flat, \tag{4.2.5}$$

where $\rho_k = 1/g(y_k, s_k)$ and $\mathcal{V}_k = \mathrm{id} - \rho_k y_k s_k^\flat$;

else

$P_k^{s+1} = I$;

end if

Set $s = s + 1$ and go to Step 2;

end if

Algorithm 16 A nonsmooth limited-memory Riemannian BFGS algorithm

Input: A starting point $x_1 \in \mathcal{M}$, $c_1 \in (0,1)$, $c_2 \in (c_1,1)$, $\theta_\epsilon, \theta_\delta \in (0,1)$, $\delta_1 > 0$, $\epsilon_1 \in (0, l(\mathcal{M}))$, $k = 1$, $P_1 = I$, a bound $1/\Lambda > 0$ on $\frac{y_k^\flat s_k}{y_k^\flat y_k}$ and λ on $\frac{s_k^\flat y_k}{s_k^\flat s_k}$

1: Step 1: (Set new parameters) $s = 1$, $x_k^s = x_k$ and $P_k^s = P_k$;

2: Step 2: (Descent direction) $(g_k^s, p_k^s) = Descent(x_k^s, \delta_k, c_1, \epsilon_k, P_k^s)$

3: **if** $\|g_k^s\| = 0$ **then** Stop;

4: **end if**

5: **if** $\|g_k^s\|^2 \leq \delta_k$ **then**

6: set $\epsilon_{k+1} = \epsilon_k \delta_\epsilon$, $\delta_{k+1} = \delta_k \theta_\delta$, $x_{k+1} = x_k^s$, $P_{k+1} = P_k^s$, $k = k+1$. Go to Step 1;

7: **else**

8:

$$\alpha = Line(x_k^s, p_k^s, g_k^s, P_k^s, c_1, c_2)$$

and construct the next iterate $x_k^{s+1} = R_{x_k^s}(\alpha p_k^s)$, $\xi_k \in \partial f(x_k^{s+1})$ and define $s_k := \mathcal{T}_{x_k^s \to R_{x_k^s}(\alpha p_k^s)}(\alpha p_k^s)$, $y_k := \frac{1}{\beta_{\alpha p_k^s}} \xi_k - \mathcal{T}_{x_k^s \to R_{x_k^s}(\alpha p_k^s)}(g_k)$, $s_k := s_k + \max(0, \frac{1}{\Lambda} - \frac{s_k^\flat y_k}{y_k^\flat y_k})$;

9: **if** $\frac{s_k^\flat y_k}{s_k^\flat s_k} \geq \lambda$ **then** Update

$$P_k^{s+1} = \tilde{\mathcal{V}}_k^\flat \tilde{\mathcal{V}}_{k-1}^\flat \dots \tilde{\mathcal{V}}_{k-m}^\flat \tilde{\mathcal{H}}_{k+1}^0 \tilde{\mathcal{V}}_{k-m} \cdots \tilde{\mathcal{V}}_{k-1} \tilde{\mathcal{V}}_k \tag{4.2.6}$$

$$+ \rho_{k-m} \tilde{\mathcal{V}}_k^\flat \tilde{\mathcal{V}}_{k-1}^\flat \cdots \tilde{\mathcal{V}}_{k-m+1}^\flat \tilde{\mathcal{V}}_{k-1}^\flat s_{k-m}^{(k+1)} s_{k-m}^{(k+1)} \tilde{\mathcal{V}}_{k-m+1} \cdots \tilde{\mathcal{V}}_{k-1} \tilde{\mathcal{V}}_k \tag{4.2.7}$$

$$+ \cdots + \rho_k s_k^{(k+1)} s_k^{(k+1)}, \tag{4.2.8}$$

where $\rho_k = 1/g(y_k, s_k)$, $\tilde{\mathcal{V}}_i = \mathrm{id} - \rho_k y_i^{(k+1)} s_i^{(k+1)\flat}$, $s_i^{(k+1)} \in \mathrm{T}_{x_{k+1}} \mathcal{M}$ is given by transporting s_i and likewise for $y_i^{(k+1)}$;

10: **else**

11: $P_k^{s+1} = I$;

12: **end if**

13: Set $s = s + 1$ and go to Step 2;

14: **end if**

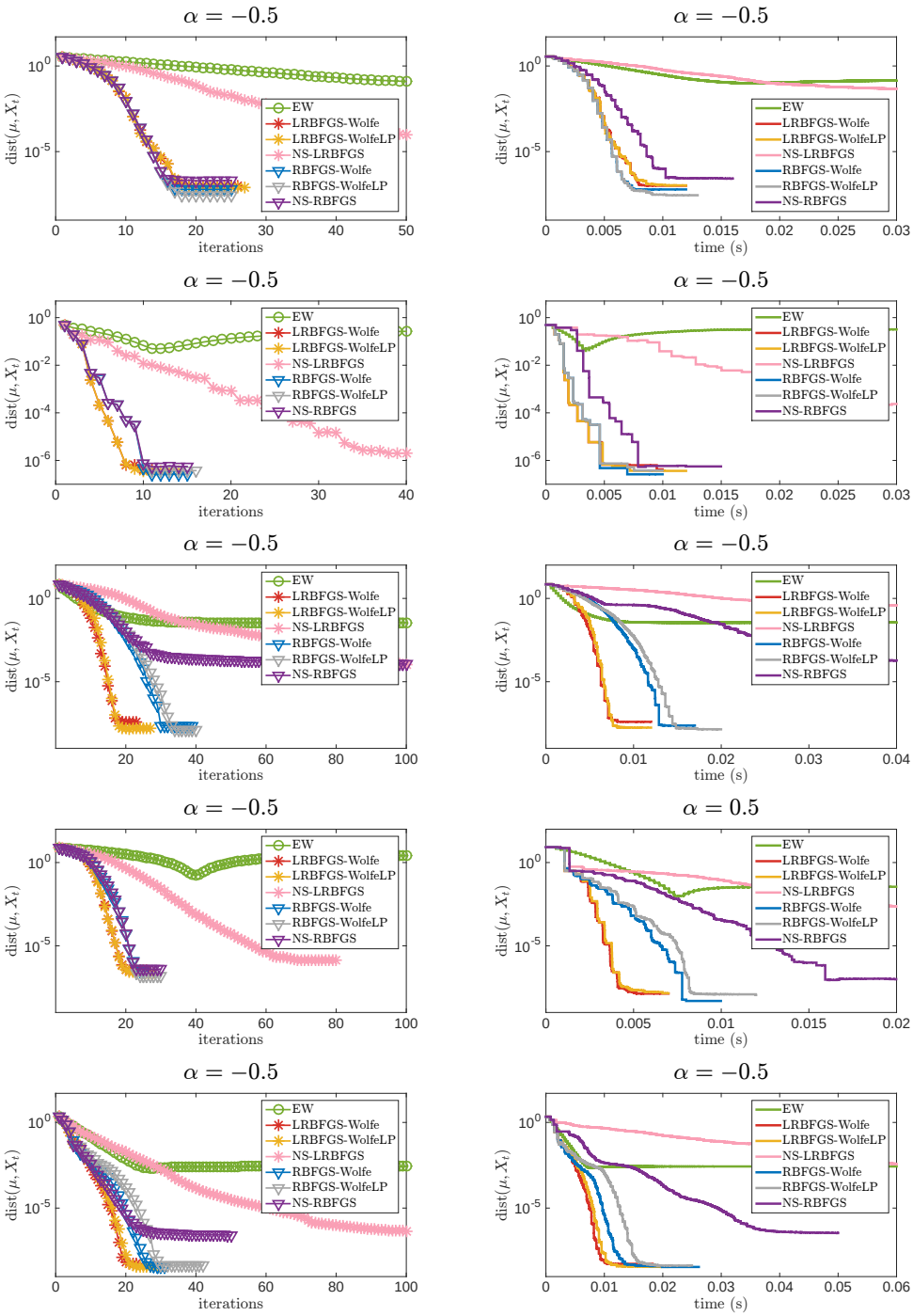


Figure 4.12: Evolution of averaged distance between current iterate and the exact LogDet α-divergence median with respect to time and iterations for $K = 100$ and $n = 3$. Row 1: A_i's belong to a small ball $B(I, r)$ centered at the identity matrix; Row 2: A_i's belong to a small ball centered at an ill-conditioned matrix; Row 3: well conditioned A_i's with 5% ill-conditioned outliers; Row 4: ill conditioned A_i's with 5% well-conditioned outliers; Row 5: A_i's are separated into 4 clusters.

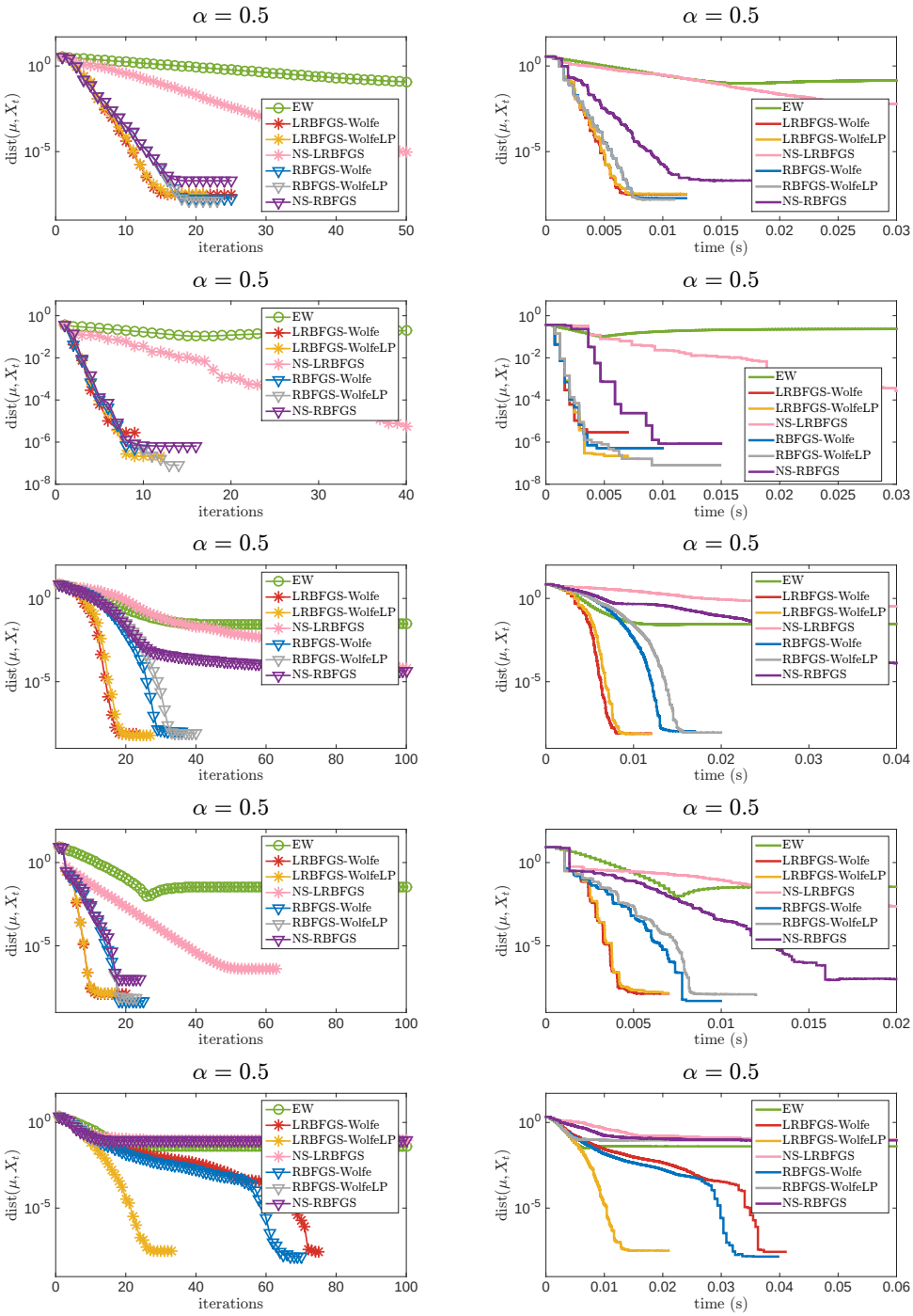


Figure 4.13: Evolution of averaged distance between current iterate and the exact LogDet α-divergence median with respect to time and iterations for $K = 100$ and $n = 3$. Row 1: A_i's belong to a small ball $B(I, r)$ centered at the identity matrix; Row 2: A_i's belong to a small ball centered at an ill-conditioned matrix; Row 3: well conditioned A_i's with 5% ill-conditioned outliers; Row 4: ill conditioned A_i's with 5% well-conditioned outliers; Row 5: A_i's are separated into 4 clusters.

CHAPTER 5

L^∞ RIEMANNIAN CENTER OF MASS COMPUTATION ON $\mathcal{S}^n_{++}$

5.1 Introduction

Given a set of SPD matrices $\{A_1, \ldots, A_K\}$, their L^∞ Riemannian center of mass—also termed minimax center in [9]—is defined as the point minimizing the maximum dissimilarity δ to the point set

$$\mu_\infty = \underset{X \in \mathcal{S}^n_{++}}{\arg\min} \max_{1 \le i \le K} \delta(A_i, X), \tag{5.1.1}$$

where δ is a distance or a divergence. This problem is also known as the smallest enclosing ball problem, the minimum enclosing ball problem, 1-center problem or the minimax optimization problem.

In the Euclidean space, finding the unique smallest enclosing ball of a finite point set $\{a_1, \ldots, a_K\}$ is a classical problem of computational geometry and a fast and simple iterative procedure has been proposed in [9]. The procedure is extended to arbitrary Riemannian manifold in [9] with a study of the convergence rate. The existence and uniqueness of the minimax center defined in (5.1.1) have been studied in [3, 4, 9]. Most recently, the SPD minimax center is used to denoise tensor images in [7].

In this chapter, we utilize the modified Riemannian quasi-Newton methods and the nonsmooth Riemannian quasi-Newton method discussed in previous chapter to compute the SPD minimax center based on the geodesic distance and the LogDet α-divergence.

The main contributions of this dissertation for the SPD minimax center computation are:

- Exploit the modified Riemannian quasi-Newton algorithms and the nonsmooth versions of Riemannian quasi-Newton algorithms for SPD minimax center computation.
- Empirically and systematically investigate the performance of proposed algorithms, and compare with the state-of-the-art procedure in [9].
- Investigate the SPD minimax center based on the log-determinant α-divergence.

5.2 Description of the SPD minmax center computation methods

In this section, we present algorithms for SPD minimax center computation.

5.2.1 The classical Arnaudon and Nielsen's algorithm

In [11], Badoiu and Clarkson proposed an efficient and simple procedure to compute the minimax center of a set of finite points $\{a_1, \ldots, a_K\}$ in Euclidean space:

- Initialize the minimax center x_1 with an arbitrary point of $\{a_1, \ldots, a_K\}$
- Iteratively update the center $x_{k+1} = x_k + (f_k - c_k)/(k+1)$, where f_k is the farthest point of set $\{a_1, \ldots, a_K\}$ to c_k.

Figure 5.1-5.3 illustrate Badoiu and Clarkson's procedure in $\mathbb{R}^2$. For each figure, because of the difficulty in seeing details in the leftmost plot near the minimizer, we zoom into the region near the minimizer. The yellow circle denotes the data point. The green mark $*$ is the initial point and the red mark $*$ is the final step. The yellow solid points are the intermediate iterates and the numbers indicate the order. The blue line illustrates the path from one step to the next. It is observed from Figure 5.1-5.3 that the sequence of iterates generated by Badoiu and Clarkson's procedure is bouncing back and forth near the minimizer.

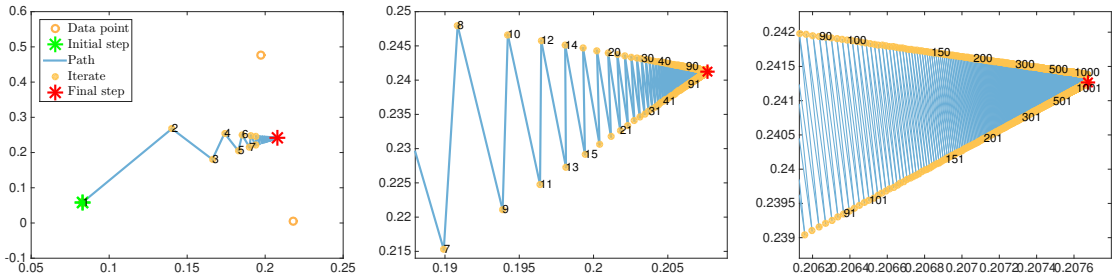


Figure 5.1: Illustration of Badoiu and Clarkson's procedure to compute the minimax center of 3 points in $\mathbb{R}^2$. From the left plot to the right plot, we zoom into the region near the minimizer.

In [9], Arnaudon and Nielsen extended Badoiu and Clarkson's procedure to arbitrary Riemannian manifolds with a study of convergence rate. In nonpositive sectional curvature manifolds as $\mathcal{S}^{\mathrm{n}}_{++}$, the convergence of this algorithm is guaranteed. We summarize Arnaudon and Nielsen's algorithm for SPD minimax center computation in Algorithm 17.

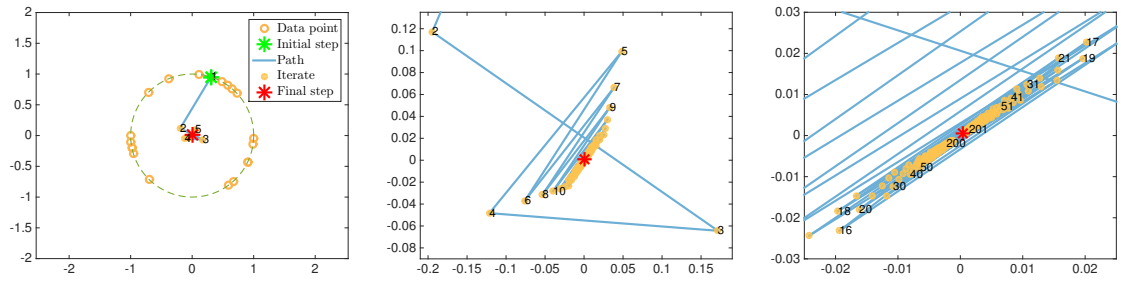


Figure 5.2: Illustration of Badoiu and Clarkson's procedure to compute the minimax center of a set of points on a circle in $\mathbb{R}^2$. From the left plot to the right plot, we zoom into the region near the minimizer.

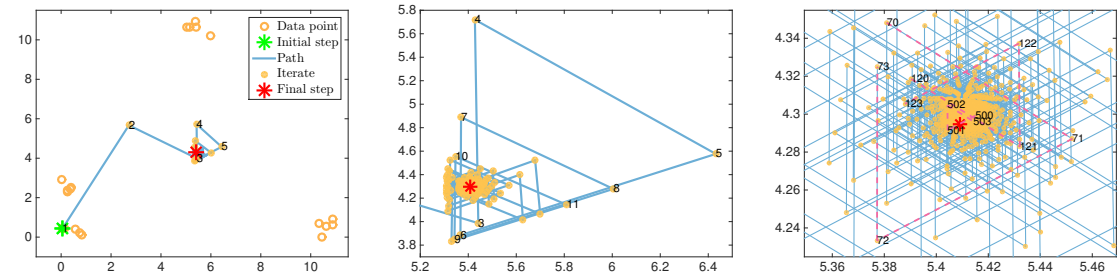


Figure 5.3: Illustration of Badoiu and Clarkson's procedure to compute the minimax center of a set of points separated into a few clusters in $\mathbb{R}^2$. From the left plot to the right plot, we zoom into the region near the minimizer.

Algorithm 17 Arnaudon and Nielsen's procedure for the SPD minimax center computation

Input: A_i; initial iterate $x_0 \in \mathcal{S}^{\mathrm{n}}_{++}$;

1: $k = 1$;

2: **repeat**

3: Obtain the farthest matrix to the current iterate

$$\bar{A} = \underset{1 \leq i \leq K}{\arg\max}\, \delta_R(x_k, A_i);$$

4: Compute $\xi_k = \mathrm{Exp}_{x_k}^{-1}(\bar{A})$;

5: $t = 1/(1+k)$;

6: $x_{k+1} = \mathrm{Exp}_{x_k}(t\xi_k)$;

7: $k = k + 1$;

8: **end(repeat)**

We extend Arnaudon and Nielsen's procedure to compute the minimax center based on the LogDet α-divergence, which is described in Algorithm 18.

Algorithm 18 Arnaudon and Nielsen's procedure for the LogDet α-divergence-based minimax center computation

Input: A_i; initial iterate $x_0 \in \mathcal{S}_{++}^{\mathrm{n}}$;

1: $k = 1$;
2: **repeat**
3: Obtain the farthest matrix to the current iterate in terms of the LogDet α-divergence

$$\bar{A} = \underset{A_i, 1 \leq i \leq K}{\arg\max}\, \delta_{\mathrm{LD},\alpha}(x_k, A_i);$$

4: Compute $\xi_k = \mathrm{Exp}_{x_k}^{-1}(\bar{A})$;
5: $t = 1/(1+k)$;
6: $x_{k+1} = \mathrm{Exp}_{x_k}(t\xi_k)$;
7: $k = k + 1$;
8: **end(repeat)**

5.2.2 Riemannian optimization methods

We exploit the modified and nonsmooth quasi-Newton algorithms discussed in Section 4.2 to compute the SPD minimax center. The detailed description of those algorithms are given in Section 4.2. Here we provide the problem-related objects.

For the SPD Riemannian minimax center problem

$$\mu_\infty = \underset{X \in \mathcal{S}_{++}^{\mathrm{n}}}{\arg\min}\, f(X), \text{ with } f : \mathcal{S}_{++}^{\mathrm{n}} \to \mathbb{R} : X \mapsto \max_{1 \leq i \leq K} \delta_R(A_i, X). \tag{5.2.1}$$

The cost function f is not differentiable if there is a tie or $X = A_i$, $i = 1, \ldots, K$. Otherwise, f is differentiable and its Riemannian gradient under the Riemannian metric is

$$\mathrm{grad}\, f(X) = -\frac{1}{\delta(A_{i_\star}, X)} \mathrm{Exp}_X^{-1}(A_{i^\star}), \tag{5.2.2}$$

where $A_{i_\star} = \arg\max_{1 \leq i \leq K} \delta(A_i, X)$.

If we replace the Riemannian geodesic distance δ_R with the LogDet α-divergence $\delta_{\mathrm{LD},\alpha}$ in problem (5.2.1), we obtain the divergence-based minimax center. Note that instead of using $\delta_{\mathrm{LD},\alpha}$, we use $\delta_{\mathrm{LD},\alpha}^2$, since we have shown in Section 3.3.2 that $\delta_{\mathrm{LD},\alpha}^2$ is jointly geodesically convex.

$$\mu_{\alpha,\infty} = \underset{X \in \mathcal{S}^{\mathrm{n}}_{++}}{\arg\min} f_\alpha(X), \text{ with } f_\alpha : \mathcal{S}^{\mathrm{n}}_{++} \to \mathbb{R} : X \mapsto \max_{1 \le i \le K} \delta^2_{\mathrm{LD},\alpha}(A_i, X). \tag{5.2.3}$$

The cost function f_α is not differentiable if there is a tie. Otherwise, f_α is differentiable and its Riemannian gradient under the Riemannian metric is

$$\operatorname{grad} f_\alpha(X) = \frac{2}{1-\alpha}\{X(\frac{1-\alpha}{2}A_{i_\star} + \frac{1+\alpha}{2}X)^{-1}X - X\}, \tag{5.2.4}$$

where $A_{i_\star} = \arg\max_{1 \le i \le K} \delta_{\mathrm{LD},\alpha}(A_i, X)$.

5.3 Numerical experiments

In this section, we compare the performances of different algorithms on various data sets, including the Arnaudon and Nielsen's procedure (AN) in Algorithm 17, the modified RBFGS for partly smooth functions (RBFGS-WolfeLP) in Algorithm 12, the modified LRBFGS for partly smooth functions (LRBFGS-WolfeLP) in Algorithm 13, the nonsmooth RBFGS in Algorithm 15 (NS-RBFGS), the nonsmooth LRBFGS in Algorithm 16 (NS-LRBFGS), and the Riemannian gradient sampling algorithm (RGS) in [46, section 7.2].

For simplicity of notation, throughout this section we denote the number, dimension, and condition number of the matrices by K, n, and κ respectively. Regarding the parameter setting, we set Wolfe parameter $c_1 = 10^{-4}$ and $c_2 = 0.999$. For the modified RBFGS and LRBFGS, ϵ and J are set to be 10^{-6} and $2d$, where $d = n(n+1)/2$ is the dimension of $\mathcal{S}^{\mathrm{n}}_{++}$. The parameters for the nonsmooth RBFGS and LRBFGS are set as follows: $\epsilon_1 = 10^{-2}$, $\delta_1 = 10^{-4}$, $\theta_\epsilon = 10^{-2}$, $\theta_\delta = 10^{-2}$, $\lambda = 10^{-2}$, $\Lambda = 10^2$. For two versions of LRBFGS, we take $m = 2$. For the RGS, we take $l = d + 1$. Unless otherwise specified, our choice of the initial iterate is an arbitrary point of $\{A_1, \ldots, A_K\}$. We run the algorithms until they reach their highest accuracy. This allows the algorithms to reach the noise floor after enough iterations.

All experiments are performed on the Florida State University HPC system with 24 Intel(R) Xeon(R) CPU E5-2680 v3 processor 2.5GHz. All the experiments are carried out using C++, compiled with gcc-4.7.x.

5.3.1 Comparison of performances between different algorithms for SPD Riemannian minimax center computation

As a first test, we investigate the performance of all 6 algorithms for SPD Riemannian minimax center computation on datasets with different distributions as in Section 4.3.1. We observed that the performance algorithms is influenced by the distributions of datasets. For each setting of parameters, 50 random runs with the same seeds are used. The test datasets $\{A_1, \ldots, A_K\}$ are constructed as follows.

I. Generate 100 data matrices that belong to a small ball $B(I, r)$ of radius r centered at the identity matrix I. More specifically, $A_i's$ are generated as follows:

   a. Generate symmetric matrices $\eta_i \in \mathrm{T}_I(\mathcal{S}^{\mathrm{n}}_{++})$;

   b. Normalize $\eta_i = \eta_i/\|\eta_i\|$;

   c. $A_i = \mathrm{Exp}_O(t\eta_i)$, $t \in (0, 1)$.

II. Generate 100 data matrices that belong to a small ball $B(O, r)$, where O is a random ill-conditioned matrix with condition number around 10^5.

III. Generate 95 well-conditioned random matrices with condition number less than 10, and add 5 ill-conditioned random matrices with condition number around 10^5 as outliers.

IV. Generate 95 ill-conditioned random matrices with condition number around 10^5, and add 5 well-conditioned random matrices with condition number around 1 as outliers.

V. Generate 100 random matrices that are separated into 4 clusters, and each cluster contains 25 matrices. More specifically, $A_i's$ are generated as follows:

   a. Generate 4 random matrices O_c that are away from each other, and the between-cluster distance is between 5 and 100.

   b. For each O_c, generate 25 random matrices that belong to $B(O_c, r)$, and the within-cluster distance is between 10^{-2} and 10^{-3}.

Figure 5.4-5.6 show the performance results of different algorithms tested on small dimension matrices with K = 100 and n = 3. In Figure 5.4, the data matrices belong to a ball $B(O, r)$ with radius $r = 0.1$. That is, $\delta(A_i, O) \leq 0.1$ for $i = 1, \ldots, K$. The center O is the identity matrix for the two plots on the top row and O is an ill-conditioned matrix with $\kappa = 10^5$ for the two plots on the bottom row. When the data matrices are centered at the identity matrix, the condition

number of matrices is between 1 and 10^3. We observe that the state-of-the-art AN procedure is outperformed by the considered Riemannian optimization algorithms in terms of number of iterations and computation time. We can also observe that the AN procedure converge very fast in the early steps, but slows down after a number of steps. It is also observed that even though RGS requires the least number of iterations per unit of accuracy required, it requires much more computation time than the other Riemannian optimizations due to its cost in solving the quadratic programming problem and a number of gradient evaluations in each iteration. In this case, the modified LRBFGS is the winner in terms of computation time. When the data matrices are centered at an ill-conditioned matrix, the condition number of data matrices is between 10^4 and 10^6. It is observed that the modified LRBFGS failed to achieve a high accuracy, and the modified RBFGS performs the best.

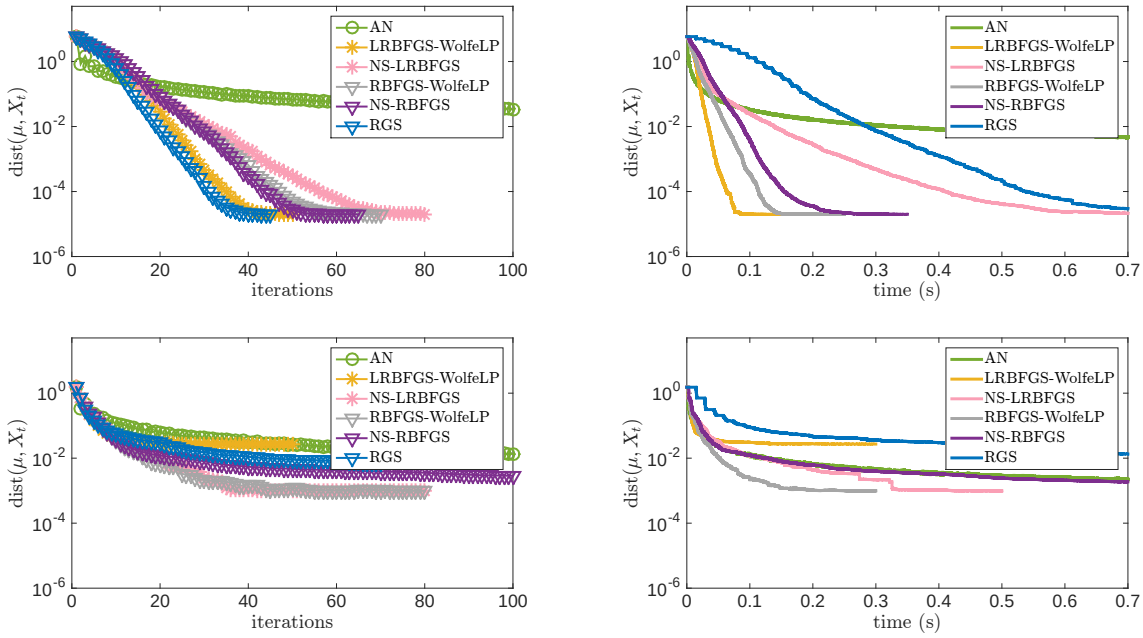


Figure 5.4: Evolution of averaged distance between current iterate and the exact Riemannian minimax center with respect to time and iterations for $K = 100$, $n = 3$. Top row: A_i's belong to a small ball $B(I, r)$ centered at the identity matrix; Bottom row: A_i's belong to a small ball centered at an ill-conditioned matrix.

Figure 5.5 reports the results tested on datasets in presence of outliers. For the two plots on the top row, the dataset contains 95 well-conditioned matrices with $1 \leq \kappa(A_i) \leq 2$ and 5 ill-conditioned

outliers with $10^4 \leq \kappa \leq 10^6$. We observe that the modified RBFGS requires the least number of iterations and computation time per unit of accuracy required. For the two plots on the bottom row, the dataset contains 95 ill-conditioned matrices with $10^4 \leq \kappa(A_i) \leq 10^6$ and 5 well-conditioned outliers with $1 \leq \kappa(A_i) \leq 2$. It is shown that the AN procedure converges very fast in the early steps, but is outperformed by the modified RBFGS later. In both cases, the modified LRBFGS cannot achieve a high accuracy.

Figure 5.6 reports the results tested on dataset in which matrices are clustered into 4 groups. Each cluster contains 25 matrices. The modified RBFGS is clearly the winner in terms of number of iterations and computation time.

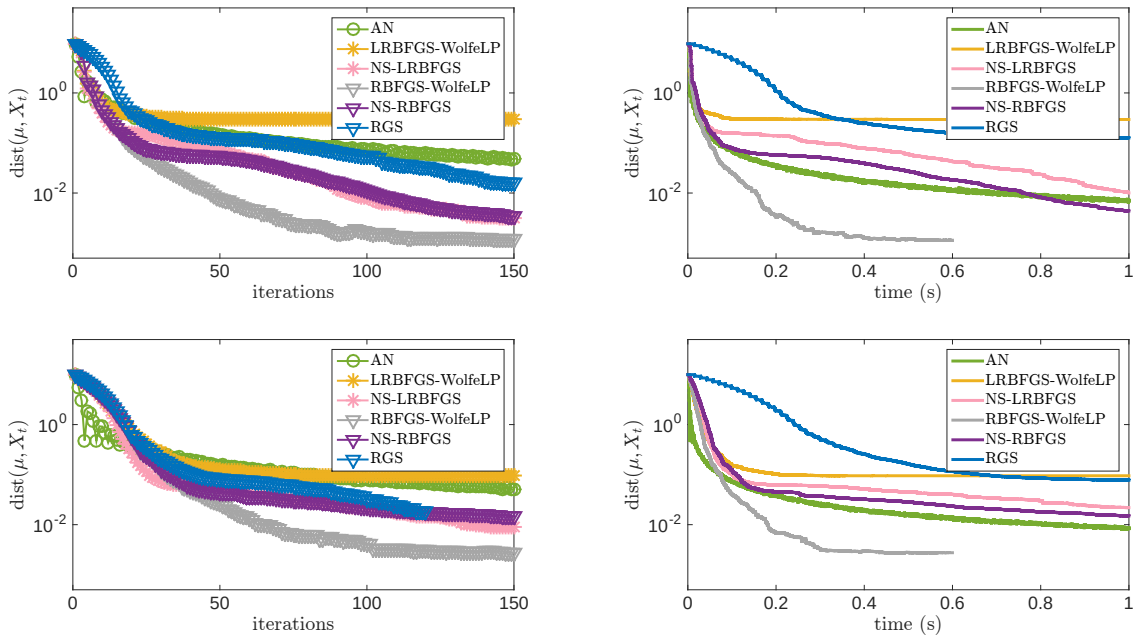


Figure 5.5: Evolution of averaged distance between current iterate and the exact Riemannian minimax center with respect to time and iterations for $K = 100$ and $n = 3$. Top row: well conditioned A_i's with 5% ill-conditioned outliers; Bottom row: ill conditioned A_i's with 5% well-conditioned outliers.

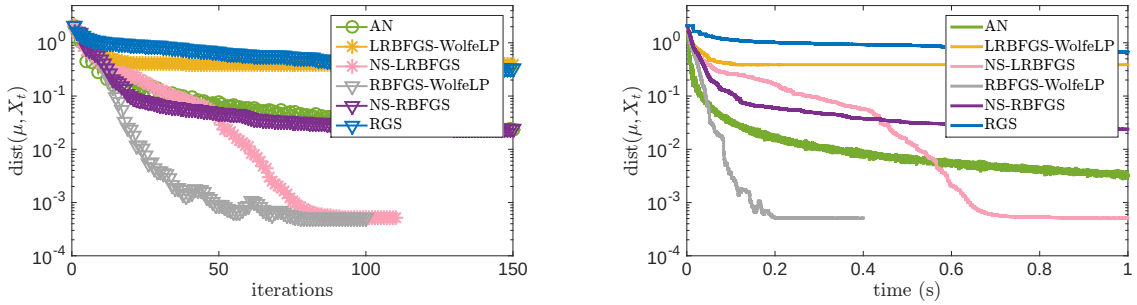


Figure 5.6: Evolution of averaged distance between current iterate and the exact Riemannian minimax center with respect to time and iterations for $K = 100$ and $n = 3$. A_i's are separated into 4 clusters.

5.3.2 Comparison of performances between different algorithms for SPD LogDet α-divergence minimax center computation

As a second test, we investigate the performance of 5 algorithms for SPD LogDet α-divergence median computation. Different values of α are considered. We use the same datasets as in Section 5.3.1.

Figure 5.7-5.9 report the results of tests conducted on datasets with $K = 100$, $n = 3$ and $\alpha = 0$. It is shown that the AN procedure fails to converge in a large number of numerical experiments for $\alpha = 0$. The performance of considered Riemannian optimization algorithms is very similar as what we observed in the computation of Riemannian minimax center in previous section. The modified RBFGS is still the winner, while its limited-memory version fails to converge.

More experiment results for $\alpha = -0.5$ and $\alpha = 0.5$ are given in Figure 5.10 and Figure 5.10.

5.3.3 Comparison between SPD Riemannian means, medians and minimax centers

In this experiment, we use tensor data, i.e., 3×3 SPD matrices, to investigate the impact of outliers on different Riemannian centers, including the mean, median, and minimax center. We can observe that the outliers have the minimum impact on the median. However, the minimax center is almost determined by the outliers.

A tensor can be visualized as an ellipsoid, whose axes point along the eigenvectors and the lengths of the axes are given by the corresponding eigenvalues. For tensor visualization, we resort

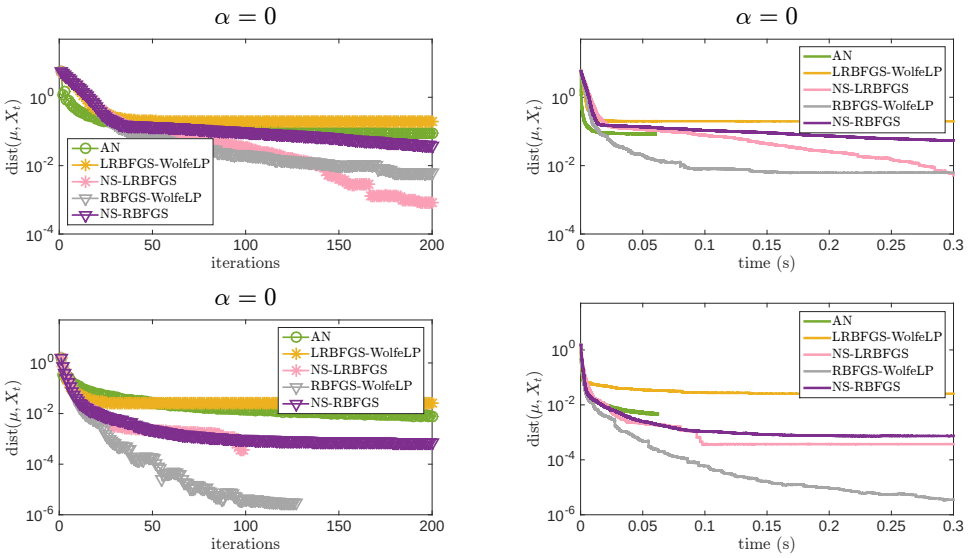


Figure 5.7: Evolution of averaged distance between current iterate and the exact LogDet α-divergence minimax center with respect to time and iterations for $K = 100$ and $n = 3$. Row 1: A_i's belong to a small ball $B(I, r)$ centered at the identity matrix; Row 2: A_i's belong to a small ball centered at an ill-conditioned matrix.

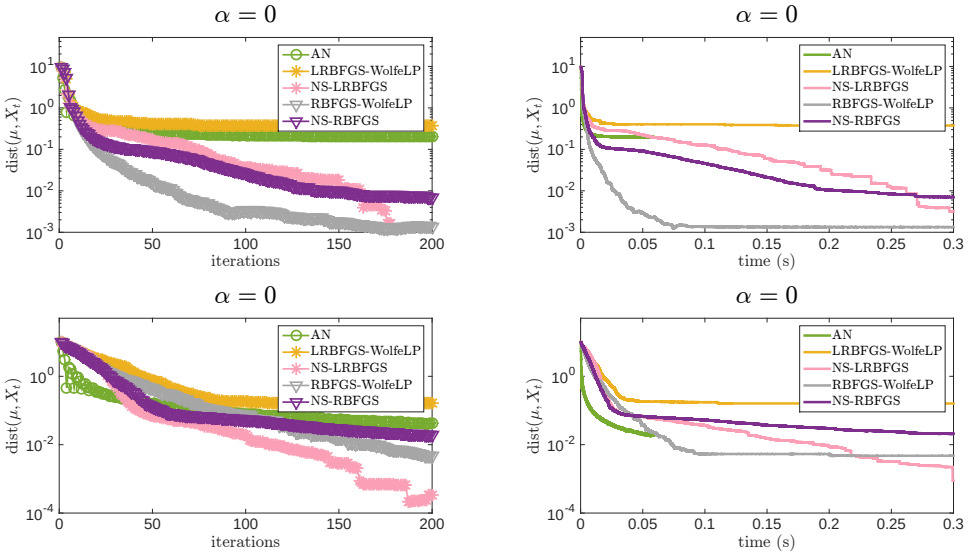


Figure 5.8: Evolution of averaged distance between current iterate and the exact LogDet α-divergence minimax center with respect to time and iterations for $K = 100$ and $n = 3$. Top row: well conditioned A_i's with 5% ill-conditioned outliers; Bottom row: ill conditioned A_i's with 5% well-conditioned outliers.

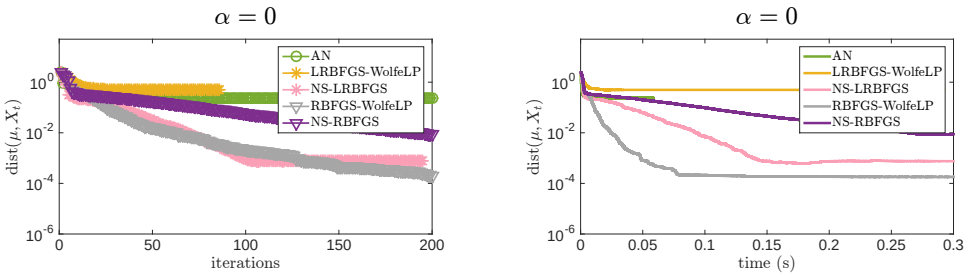


Figure 5.9: Evolution of averaged distance between current iterate and the exact LogDet α-divergence minimax center with respect to time and iterations for $K = 100$ and $n = 3$. A_i's are separated into 4 clusters.

to the fanDTasia ToolBox[1] developed by [14]. The tensors are colored based on the direction of the major eigenvector, i.e., eigenvector associated with the largest eigenvalue. The color brightness is a measure of tensor anisotropy. That is, more brightness implies more anisotropy.

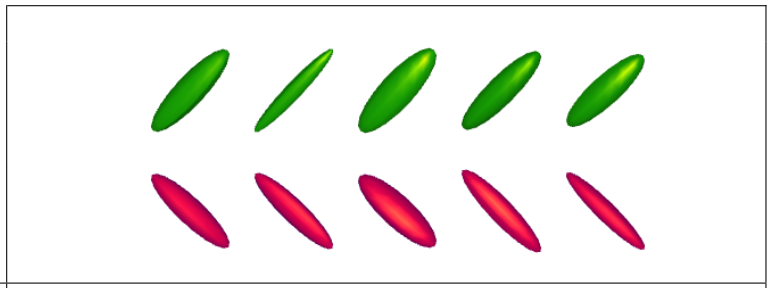
For each test, we generate 50 random 3×3 SPD matrices as following:

- Generate random orthonormal matrix O, whose columns are eigenvectors for A_i. Note that we use the same eigenvectors O for each A_i.
- Generate eigenvalues for each A_i with $D_i = (\lambda_1, \lambda_2, \lambda_3) \sim \mathcal{N}(\bar{\lambda}, \sigma^2 I)$.
- Compute $A_i = OD_iO^T$.
- Add Gaussian noise to the Cholesky factor of $A_i = L_iL_i^T$, i.e., $\tilde{L}_i = L_i + E_i$
- Compute $A_i = \tilde{L}_i\tilde{L}_i^T$.

Dataset I. For the original data tensors, we take mean $\bar{\lambda} = (4, 1, 1)$ and standard deviation $\sigma = 0.2$. We generate outliers whose eigenvalues follow the same distribution as the original data but the major eigenvector is perpendicular to that of the original tensors. The original dataset contains 50 tensors. We compute the Riemannian medians and means of the tensor dataset with 0, 1, 5, 10, 25, and 50 outliers. (When 50 outliers are added to the dataset, they are not outliers anymore.) The results are displayed in Table 5.1. Shown in the top row of Table 5.1 are 5 samples of the original dataset (green) and 5 samples of the outliers (red). The resulting means and medians are colored in yellow.

[1] https://www.mathworks.com/matlabcentral/fileexchange/26997-fandtasia-toolbox

Table 5.1: Comparison of the Riemannian mean, median and minimax center for 3×3 SPD matrices based on the geodisic distance δ_R. Shown in the top row are 5 samples of the original dataset (green) and 5 samples of the outliers (red). The major eigenvectors of the original tensors and the outliers are perpendicular to each other. The resulting means and medians are colored in yellow.



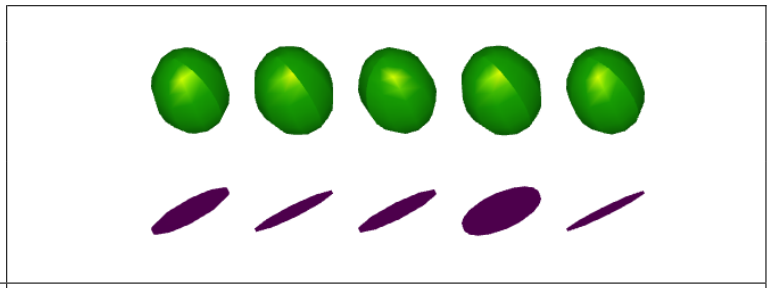
	outliers	0	1	5	10	25	50
δ_R	Mean						
	Median						
	Circumcenter						

Dataset II. In the second test, we generate tensors in a similar way. We generate 50 well-conditioned tensors with mean $\bar{\lambda} = (5, 4, 4)$ and standard deviation $\sigma = 0.2$. Then we generate ill-conditioned outliers with mean $\bar{\lambda} = (20, 1, 10^5)$ and the same standard deviation. The results are displayed in Table 5.2. Notice that the color of the outliers in Table 5.2 become very dark since they are ill-conditioned.

5.4 Conclusions

In this chapter, we consider computing the minimax center of a collection of SPD matrices based on the Riemannian geodesic distance and the log-determinant α-divergence. We exploit the modified version and the nonsmooth version of Riemannian quasi-Newton algorithms to handle this computational task. We empirically and systematically investigate the performance of proposed Riemannian quasi-Newton algorithms and compare with the state-of-the-art Arnaudon and Nielsen's procedure and Riemannian gradient sampling. We examine the performance results of

Table 5.2: Comparison of the Riemannian mean, median and minimax center for 3×3 SPD matrices based on the geodisic distance δ_R. Shown in the top row are 5 samples of the original dataset (green) and 5 samples of the outliers (red). The original tensors are well-conditioned with condition number 2, while the outliers are ill-conditioned with condition number 105. The resulting means and medians are colored in yellow.



	outliers	0	1	5	10	25	50
δ_R	Mean						
	Median						
	Circumcenter						

compared algorithms on various datasets with different distributions. Our numerical experiments provide empirical guidelines to choose between various methods.

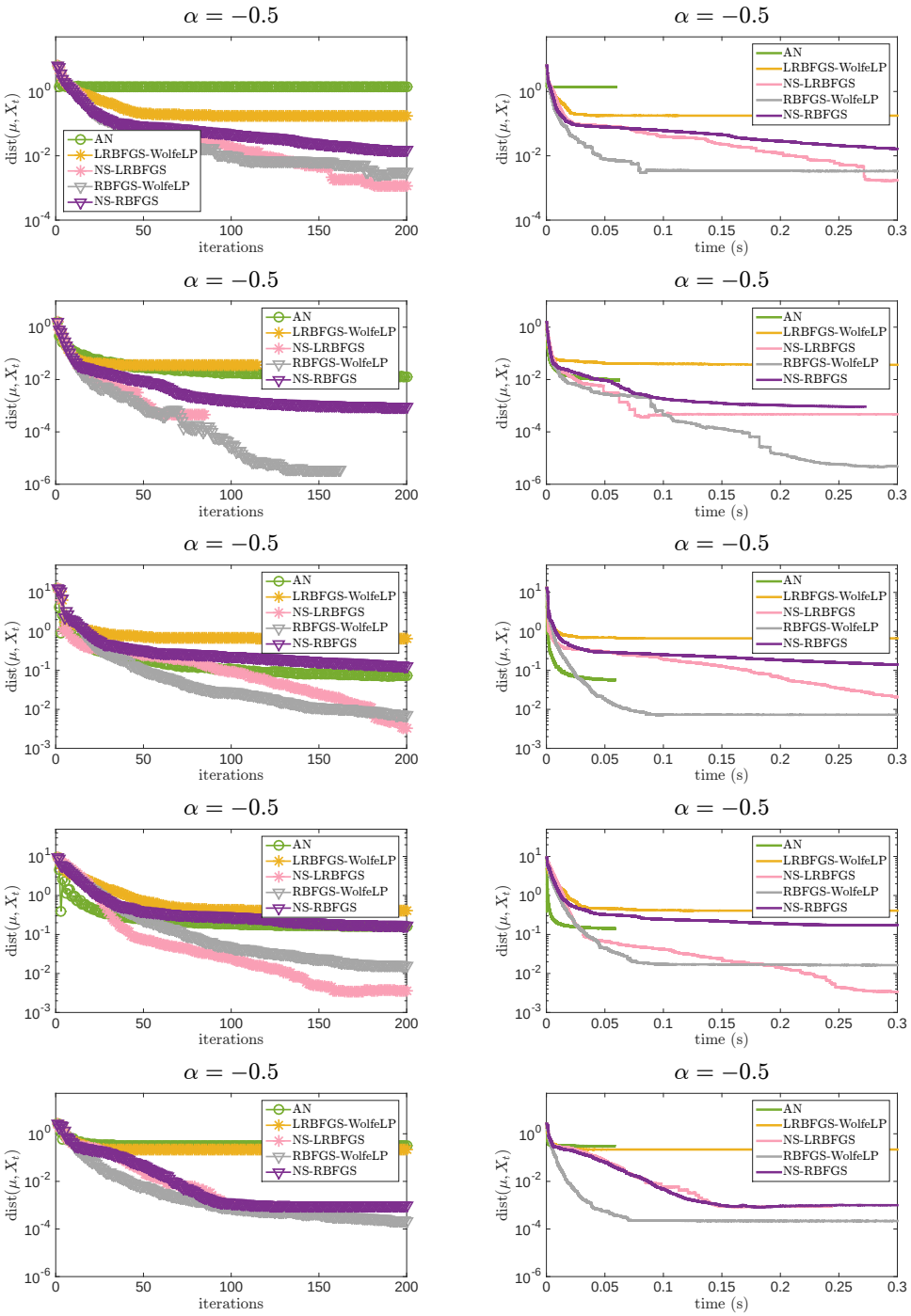


Figure 5.10: Evolution of averaged distance between current iterate and the exact LogDet α-divergence minimax center with respect to time and iterations for $K = 100$ and $n = 3$. Row 1: A_i's belong to a small ball $B(I, r)$ centered at the identity matrix; Row 2: A_i's belong to a small ball centered at an ill-conditioned matrix; Row 3: well conditioned A_i's with 5% ill-conditioned outliers; Row 4: ill conditioned A_i's with 5% well-conditioned outliers; Row 5: A_i's are separated into 4 clusters.

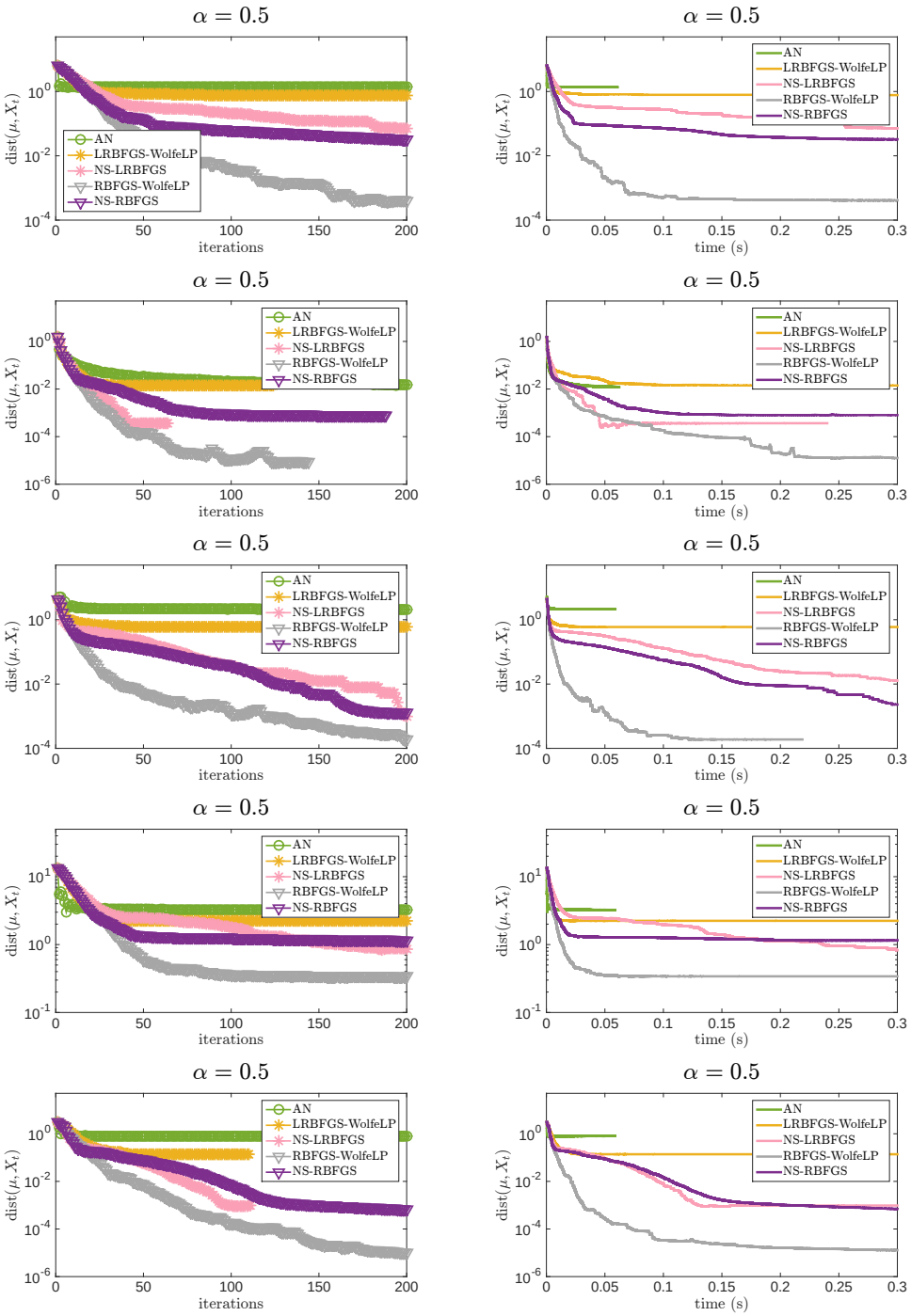


Figure 5.11: Evolution of averaged distance between current iterate and the exact LogDet α-divergence minimax center with respect to time and iterations for $K = 100$ and $n = 3$. Row 1: A_i's belong to a small ball $B(I, r)$ centered at the identity matrix; Row 2: A_i's belong to a small ball centered at an ill-conditioned matrix; Row 3: well conditioned A_i's with 5% ill-conditioned outliers; Row 4: ill conditioned A_i's with 5% well-conditioned outliers; Row 5: A_i's are separated into 4 clusters.

CHAPTER 6

APPLICATIONS

In this chapter, we present two applications that require averaging SPD matrices, and compare the performance of different averaging techniques studied in Chapter 2-4. In Section 6.2, we revisit the Electroencephalography (EEG) classification problem using the Minimum Distance to Mean classifier in [12, 55, 67], which is a supervised classification problem. In Section 6.3, we focus on the problem of unsupervised clustering. We evaluated the performance of different variants of K-means clustering on real-life data.

Main contributions.

- We provide an assessment of different averaging techniques in denoising, supervised classification and unsupervised clustering applications. We evaluate performance in two aspects: accuracy and computation time.
- We consider the problem of structure tensor image denoising.
- For the EEG classification problem studied in [55,67], we provide more efficient algorithms to compute the Karcher mean and α-divergence-based mean while achieving the same accuracy as in the literature. (We have shown in Chapter 2 and 3 the superior performance of our proposed algorithms compared with the state-of-the-art method in the literature. For the dissertation, we focus on comparing the classification accuracy with literature [55, 67]. In future work, we will implement their algorithms in C++ and compare the clustering time.)
- We propose to equip the minimum distance to mean classifier with Riemannian medians, which yields slightly higher accuracy than the means without sacrificing computation time for EEG classification. Another surprisingly good result is the performance of the Jeffrey divergence and its related mean (i.e., the arithmetic-harmonic mean). This pair achieves a satisfactory accuracy at a very low computational cost.
- We test different variants of K-means clustering on the KRH-TIPS2 dataset [66] and Virus dataset [61], and provide an illustration of trade-off between clustering accuracy and computation time.

- We contribute a C++ toolbox to estimate different means and medians of a set of SPD matrices, and provide users with options to select the best suited one based on their use cases.

6.1 Application I: structure tensor image denoising

In this section, we consider an application of different averaging techniques studied in Chapter 2-5 in structure tensor image denoising. This problem has been studied in [7, 21]. We revisit this problem as more averaging techniques and better algorithms are available.

6.1.1 Structure tensor image denoising

In image processing, structure tensor [38] has been widely used to represent the local orientation and edge information of the image, e.g., see [33, 41, 58, 70, 79]. Given a 2D scalar-valued $I : \Omega \subset \mathbb{Z}^2 \to \mathbb{R}$, its associated structure tensor is defined as

$$f(i,j) = G_\sigma \cdot \begin{bmatrix} I_i^2 & I_i I_j \\ I_i I_j & I_j^2 \end{bmatrix}, \tag{6.1.1}$$

where $I_i = \frac{\partial I(i,j)}{\partial i}$, $I_y = \frac{\partial I(i,j)}{\partial j}$, and G_σ is a Gaussian function with standard deviation σ. A structure tensor image is a spatial structured matrix field

$$f : \Omega \to \mathcal{S}_{++}^{\mathrm{n}}\,. \tag{6.1.2}$$

That is, at each pixel is an SPD matrix. Figure 6.1 gives an example of structure tensor image of a 2D image. Each pixel (i, j) is a 2×2 SPD matrix, which is visualized as an ellipse. In this dissertation, we focus on 2D images. In particular, we resort to the diffc Toolbox `https://www.mathworks.com/matlabcentral/fileexchange/5103-toolbox-diffc` for the structure tensor image computation and visualization.

6.1.2 Experiment results

In this section, we evaluate the performance of different averaging techniques for structure tensor denoising. The experiments are designed in a similar way as in [7]. Given a structure tensor image $f(i, j) : \Omega \to \mathcal{S}_{++}^2$, we generate a noisy tensor image $\tilde{f}$ by replacing the pixel elements by an outlier tensor with a given probability Pr. Denoising is done by averaging matrices in the neighborhood of each pixel as shown in Figure 6.2. That is,

$$Avg(\tilde{f})(i,j) = Average(\tilde{f}(u,v) : (u,v) \in \mathcal{N}(i,j)) \tag{6.1.3}$$

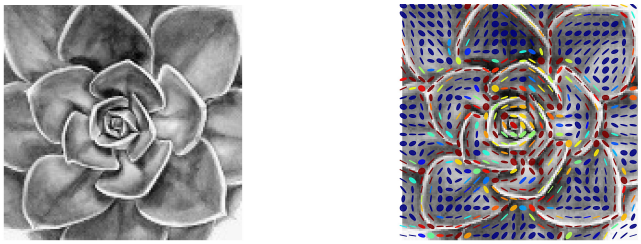


Figure 6.1: Example of structure tensor image. Left: original image; Right: corresponding structure tensor image.

where $\mathcal{N}(i,j)$ is the neighborhood pixels of (i,j). In our experiment, we take $\mathcal{N}(i,j)$ as a 3×3 square neighborhood centered at (i,j).

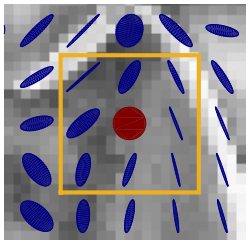


Figure 6.2: Denoising is done by averaging matrices in the neighborhood of each pixel.

Table 6.2 summarizes the notations, averaging techniques considered and the algorithms used in our experiments. In order to quantitatively evaluate the denoising effect of different averaging techniques, we use the Mean Riemannian Error (MRE) proposed in [7] as a measure of error, defined as

$$MRE = \frac{1}{|\Omega|} \sum_{(i,j)\in\Omega} \delta_{\mathrm{R}}(f(i,j), Avg(\tilde{f})(i,j)), \tag{6.1.4}$$

where δ_{R} denotes the Riemannian distance. The MRE measures the average Riemannian distance between the original tensor image and the denoised tensor image.

Figure 6.3 displays the denoising performance for different averaging techniques. Figure 6.4-6.6 show some denoised images. We observe that when $Pr = 0.02$ and $Pr = 0.1$, the Riemannian median yields the lowest MRE among all the averaging techniques. The LogDet α-divergence-based median yields better performance than the LogDet α-divergence-based mean. When $Pr = 0.5$, it is shown that the Riemannian minimax center yields better performance than the Riemannian mean

Table 6.1: A summary of notations, averaging techniques considered and the algorithms used in the structure tensor denoising. One can refer to Table 3.3 for formulas of distances/divergences.

Notation	Averaging technique	Algorithm
E.	δ_{E}-mean	Closed form
LE.	δ_{LE}-mean	Closed form
J-Div.	δ_{J}-mean	Closed form
R.	δ_{R}-mean	Algorithm 7
R.-Med.	δ_{R}-median	Algorithm 13
R.-MM.	δ_{R}-minimax center	Algorithm 12
α-Div.	$\delta_{\mathrm{LD},\alpha}$-mean	Algorithm 7
α-Med.	$\delta_{\mathrm{LD},\alpha}$-median	Algorithm 13
α-MM.	$\delta_{\mathrm{LD},\alpha}$-minimax center	Algorithm 12

and median. The LogDet α-divergence gives the lowest MRE. The value of α is set to 0.1 through cross validation. Figure 6.7 displays the MRE for different values of α.

6.2 Application II: EEG classification based on the minimum distance to mean classifier

6.2.1 EEG classification

Electroencephalography (EEG) system is widely used to record brain signals in Brain-Computer Interface (BCI) devices. For EEG signal processing, approaches based on covariance matrices have demonstrated good performance [94,95], where EEG signals are represented by covariance matrices. In this section, we consider an EEG classification problem discussed in [12,55,67]. In a steady-state visual evoked potential (SSVEP) experiment, blinking LEDs with different frequencies are placed at different locations in the visual of a subject. The subject is either asked to focus on one specific blinking LED or to focus on a location without LED. The blinking stimulus induced oscillations in the brain, which are recorded by EEG. The task is to decide which LED the subject is staring at based on the recorded EEG signals. In [55, 67], the Minimum Distance to Mean (MDM) classifier has been considered to handle this classification task. The MDM classifier requires the computation of cluster centers and a measure of distance, which is summarized in Algorithm 19. We revisit this problem since we have better algorithms to average SPD matrices and have more choices of cluster centers and dissimilarity measure.

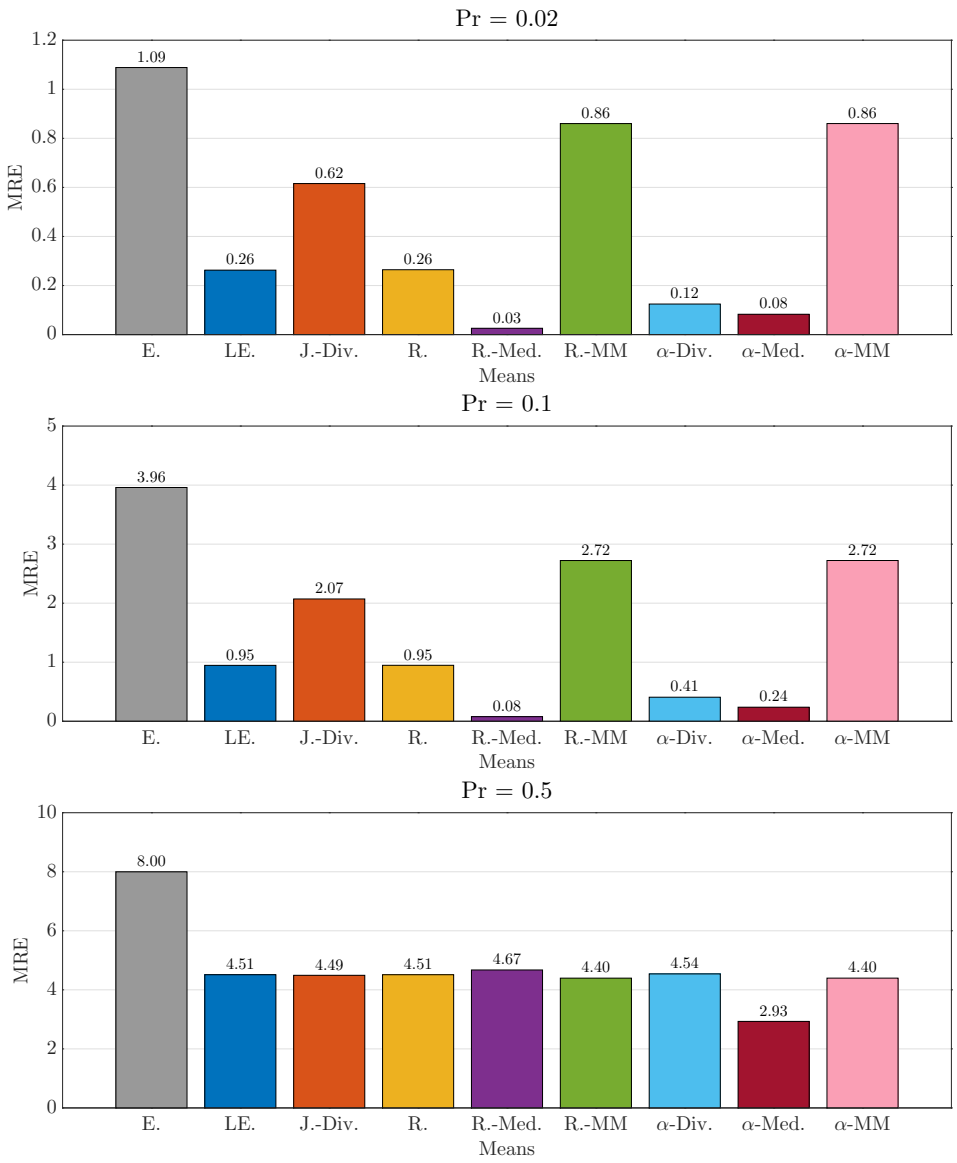


Figure 6.3: Average Mean Riemannian Error (MRE) over 10 experiments for the structure tensor image denoising.

Our experiment is conducted on the same datasets as in [12,55], where the data was obtained in a SSVEP experiment. EEG signals are recorded from 12 subjects. Each subject is presented with three LEDs blinking respectively at 13Hz, 17Hz and 21Hz. This is thus a 4-classes classification problem including 3 frequencies and one resting class. In a session, 32 trials are recorded and each class contains 8 trails. For each subject, the number of sessions recorded varies from 2 to 5. As

Algorithm 19 Minimum Distance to Mean Classifier [55]

Input: training set $\{(X_i, y_i)\}_{i=1}^N$, where $X_i \in \mathcal{S}_{++}^{\mathrm{n}}$ and $y_i \in \{1, 2, \ldots, C\}$ is the class label, an unlabelled data point $X \in \mathcal{S}_{++}^{\mathrm{n}}$.

1: **for** $k = 1, \ldots, C$ **do**
2: Compute the center of each class: $\bar{\Sigma}^{(k)} = \mu(\{X_i | y_i == k\})$;
3: **end for**
4: Compute the class label of X: $k^\star = \arg\min \delta(\bar{\Sigma}^{(k)}, X)$;
5: Return $k^\star$.

in [12, 55], a test set contains 32 trials and the remaining trials are used for training. The EEG signals are already available in covariance matrix representations. We refer the reader to [55] for more details regarding the data information and experimental protocol.

Table 6.2 summarizes all the distances/divergences and cluster centers considered in our experiments. When the closed form expression of the center is not available, LRBFGS in Algorithm 7 with $m = 2$ is applied to compute the center, and the arithmetic-harmonic mean is used as the initial iterate. We note that it is not necessary to compute the cluster center at a high accuracy. So the stopping criterion for LRBFGS is $\|\operatorname{grad} f(X)\|/\|\operatorname{grad} f(X_0)\| < 10^{-3}$. All experiments are performed on the Florida State University HPC system using Intel(R) Xeon(R) CPU E5-2670 2.60GHz. To obtain sufficiently stable timing results, an average time is taken of several runs for a total runtime of at least 1 minute for each classification task.

Table 6.2: A summary of cluster centers and distances/divergences considered in the experiments. One can refer to Table 3.3 for formulas of distances/divergences.

Notation	Distance/Divergence	Center	Closed form
Euc.	Euclidean distance	δ_{E}-mean	Yes
LogEuc.	Log-Euclidean distance	δ_{LE}-mean	Yes
Ind.	Riemannian distance	inductive mean	Yes
J-Div.	Jeffrey divergence	δ_{J}-mean	Yes
Rie.	Riemannian distance	δ_{R}-mean	No
α-Div.	LogDet α-divergence	$\delta_{\mathrm{LD},\alpha}$-mean, $\alpha = 0.6$	No
S-α-Div.	Symmetric LogDet α-divergence	$\delta_{\mathrm{S1LD},\alpha}$-mean, $\alpha = 0.9$	No
α-Med.	LogDet α-divergence	$\delta_{\mathrm{LD},\alpha}$-median, $\alpha = 0.5$	No
S-α-Med.	Symmetric LogDet α-divergence	$\delta_{\mathrm{S1LD},\alpha}$-median, $\alpha = 0.7$	No

6.2.2 Experiment results

Table 6.3 reports the classification accuracy on the test set and computation time for each subject using different pairs of distances and centers. The computation time is the average time required to compute the cluster center of each cluster in the training set. Each row of the table corresponds to one subject, and the last row corresponds to the average results over 12 subjects. Figure 6.8 illustrates Table 6.3 graphically. Figure 6.9 displays the average results over 12 subjects.

We observe that for the same distance/divergence, the median yields higher accuracy than the mean. The median based on the LogDet α-divergence with $\alpha = 0.5$ yields the best accuracy (82.30%), whose computation time is also very competitive. The value of α is obtained by testing different values of α from 0 to 0.9, and $\alpha = 0.5$ gives the best results. Figure 6.10 and Table 6.4 display the average accuracy and computation time for different values of α. A surprisingly good result is the performance of the mean based on the Jeffrey divergence, which is actually the arithmetic-harmonic mean, see Lemma 3.3.1. That is, the arithmetic-harmonic mean is taken as the center for each cluster, and the Jeffrey divergence is used as the distance measure. It has a satisfactory accuracy of 81.17%, and can be computed very efficiently. In addition, we notice that it is important to match up the distance/divergence and related mean. Table 6.5 summarizes the classification accuracy of the arithmetic-harmonic mean combined with different distance/divergences, and the Jeffrey divergence yields the best result.

6.3 Application II: K-means clustering

In this section, we evaluate the performance of different averaging techniques studied in Chapter 2-4 on unsupervised clustering task. More specifically, we test the K-means clustering on KTH-TIPS2 dataset [66] using different cluster centers and distances/divergences.

6.3.1 K-means clustering algorithm

K-means clustering (or Lloyd's algorithm [64]) is a method commonly used to partition a data set into a few groups. Given a set of K observations and the number of clusters C, K-means clustering aims to partition n observations into C clusters. The algorithm proceeds by alternating between two steps:

- Assignment step: Assign each observation to the cluster which has the closest center.

- Update step: Compute the new centers of the observations in each cluster.

The procedure is terminated until the cluster assignments no longer change or until the maximum number of iterations has been reached. We summarize K-means clustering on $\mathcal{S}_{++}^{\mathrm{n}}$ in Algorithm 20.

Algorithm 20 K-Means Clustering on $\mathcal{S}_{++}^{\mathrm{n}}$

Input: a set of SPD matrices $\mathcal{X} = \{X_i\}_{i=1}^{K}$; number of desired clusters C.

Initialization: arbitrarily choose C data matrices $\{\mu_1, \ldots, \mu_C\}$ from $\mathcal{X}$ as centers;
repeat
  for $i = 1, \ldots, K$ **do**
    for $j = 1, \ldots, C$ **do**
      Compute the distance between X_i and cluster center μ_j: $\delta(X_i, \mu_j)$;
    end for
    Assign X_i to the cluster with the nearest center $y_i = \arg\min_{1 \le j \le C} \delta(X_i, \mu_j)$;
  end for
  for $j = 1, \ldots, C$ **do**
    Compute the new center of each cluster: $\mu_j = \mu(\{X_i | y_i == j\})$;
  end for
end(repeat) convergence
Return a partitioning of $\mathcal{X}$.

6.3.2 Performance metrics

There exist several standard metrics to assess the performance of clustering algorithms. In particular, [90] presents an overview of various metrics for comparing clusterings along with their pros and cons. Of these various performance metrics, the F1-score and cluster purity are frequently used in literature, e.g., see [24, 86]. We also chose those two metrics as our performance metrics, and describe them in the next two sections.

F1-score. Suppose the ground truth clustering assignments are known, an intuitive way to assess the performance of a clustering method is counting pairs of observations that are clustered in the same way as the ground truth. Given a set of observations $\mathcal{X} = \{X_1, \ldots, X_K\}$ and pairs of observations $(X_i, X_j) \in \mathcal{X}$, we can define the true positive (TP), false positive (FP), true negative (TN), and false negative (FN) as given in Table 6.6. The precision and recall of clustering are defined, respectively, as

$$P = \frac{TP}{TP + FP} \quad \text{and} \quad R = \frac{TP}{TP + FN}. \tag{6.3.1}$$

The F1-score is the harmonic mean of precision and recall:

$$F1 = 2 \times \frac{P \times R}{P + R}, \tag{6.3.2}$$

which reaches its best value at 1 and worst 0.

Performance metric based on Normalized Mutual Information. The concept of mutual information originates from information theory and is based on the notion of entropy [29]. [69] applied the concept of entropy to clustering analysis. Assume that $\Pi = \{\pi_1, \ldots, \pi_l\}$ and $\Pi' = \{\pi'_1, \ldots, \pi'_m\}$ are two clusterings of $\mathcal{X} = \{X_1, \ldots, X_K\}$. The entropy associated with clustering Π is defined in [69], as

$$\mathcal{H}(\Pi) = -\sum_{i=1}^{l} P(i) \log_2 P(i), \tag{6.3.3}$$

where $P(i) = |\pi_i|/K$ is the probability that an observation randomly taken from $\mathcal{X}$ belongs to cluster π_i in partition Π. The mutual information between two clusterings Π and Π' is defined as

$$\mathcal{I}(\Pi, \Pi') = \sum_{i=1}^{l} \sum_{j=1}^{m} P(i,j) \log_2 \frac{P(i,j)}{P(i)P(j)}, \tag{6.3.4}$$

where $P(i,j) = |\pi_i \cap \pi'_j|/K$ is the probability that an observation belongs to cluster π_i in Π and to cluster π'_j in Π'. The mutual information $\mathcal{I}$ provides a measure to compare two clusterings. However, it is difficult to interpret since it is not bounded by a constant. A normalized version of mutual information between clusterings is introduced in [87], which is defined as

$$\mathcal{NMI}(\Pi, \Pi') = \frac{\mathcal{I}(\Pi, \Pi')}{\sqrt{\mathcal{H}(\Pi)\mathcal{H}(\Pi')}}. \tag{6.3.5}$$

The value of $\mathcal{NMI}$ is between 0 and 1. When $\Pi = \Pi'$, we have $\mathcal{NMI}(\Pi, \Pi') = 1$.

6.3.3 Experiments on real data

We evaluate the K-means clustering based on different cluster centers and distance/divergences on the KTH-TIPS2 (Textures under varying Illumination, Pose and Scale) dataset [66] and the Virus dataset [61]. The KTH-TIPS2 is a popular dataset for material recognition, which contains 4752 samples belonging to 11 different categories of materials. Each categories contains 432 samples. Figure 6.7 displays a few samples from this dataset. In our experiments, we use Region Covariance Matrices (RCMs) [43] of size 23×23 as image descriptors in the same way as [86]. The Virus

dataset contains 1500 images belonging to 15 different virus types. A few samples samples from the virus dataset are displayed in Figure 6.8. For virus images, we use RCMs of size 29×29 as descriptors.

All experiments are performed on the Florida State University HPC system using Intel(R) Xeon(R) CPU E5-2670 2.60GHz. In order to reduce the effect of initializations, we average our results over 10 runs for each clustering task. When the closed form expression of the center is not available, LRBFGS in Algorithm 7 with $m = 2$ is applied to compute the center, and the arithmetic-harmonic mean is used as the initial iterate. The stopping criterion for LRBFGS is $\| \operatorname{grad} f(X)\| / \| \operatorname{grad} f(X_0)\| < 10^{-5}$.

Table 6.9 summarizes the table notation used when reporting the results. Table 6.10 summarizes experiment results obtained on the KTH-TIPS2 dataset for each cluster center and distance/divergence, and Figure 6.11 illustrates Table 6.10 graphically. We observe that the Euclidean distance and mean (i.e., arithmetic mean) lead to efficient but poor clustering. The pair of the Log-Euclidean mean and distance is outperformed by the pair of Jeffrey divergence and its mean (i.e., the arithmetic-harmonic mean) in terms of clustering quality and time efficiency. The pair of Karcher mean and the Riemannian distance significantly improves the clustering quality which, however, requires the most computation time. The inductive mean proposed in [68] and the Riemannian distance yields similar F1-score and NMI as the Karcher mean, but approximately halves the computation time. Note that the dominant computation time (94%) for this pair is on the distance evaluation. The LogDet α-divergence and its related mean lead to the highest NMI value (57.21%) when $\alpha = 0.9$ and requires the least computation time (except for the Euclidean mean). The best F1-score (45.75%) is obtained when using the LogDet α-divergence with $\alpha = 0.9$ and its related median. In order to find the best α, we vary the values of α from -0.9 to 0.9 with the increment by 0.1. Figure 6.12 displays the results at different values of α for the LogDet α-divergence and its symmetrized version. We observe that the K-means clustering using the LogDet α-divergence and its related mean/median requires less time than that of its symmetrized version for different values of α. Moreover, the LogDet α-divergence with a well chosen value of α can lead to the best clustering quality.

Table 6.11 summarizes experiment results obtained on the virus dataset, and Figure 6.13 illustrates Table 6.11 graphically. We notice that the performance of k-means clustering on the

virus dataset is worse than that on the KTH-TIPS2 dataset. This may be caused by the fact that we loose some image information when using the Region Covariance Matrices to describe images. Among different variants of K-means clustering, the pair of LogDet α-divergence and its related mean leads to the highest F1-score and NMI, which is the same as the KTH-TIPS2 dataset.

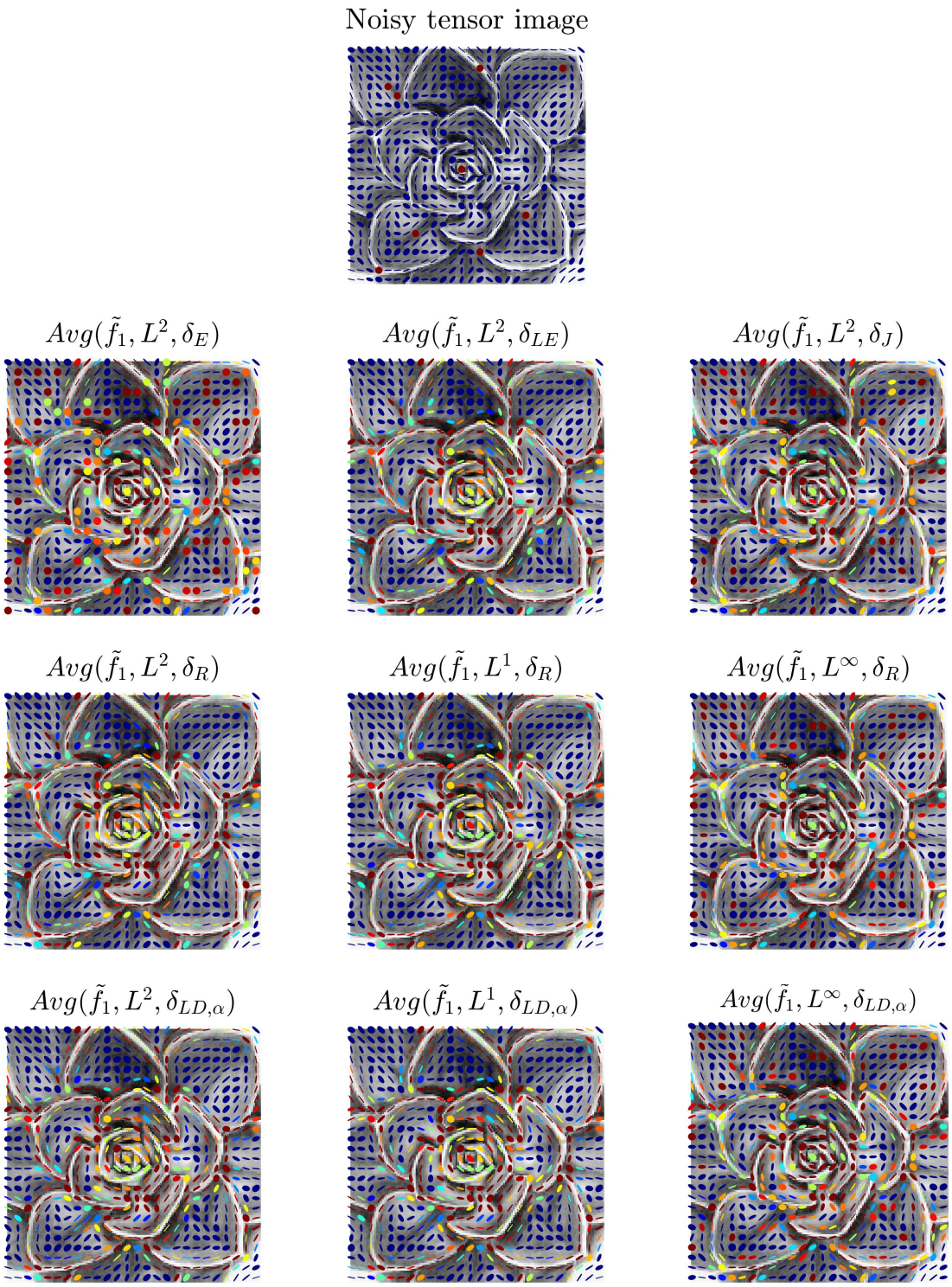


Figure 6.4: Comparison of different averaging techniques for structure tensor image denosing. First row: simulated noisy image with $Pr = 0.02$; Second-fourth row: denoised images by different averaging techniques as specified in the titles.

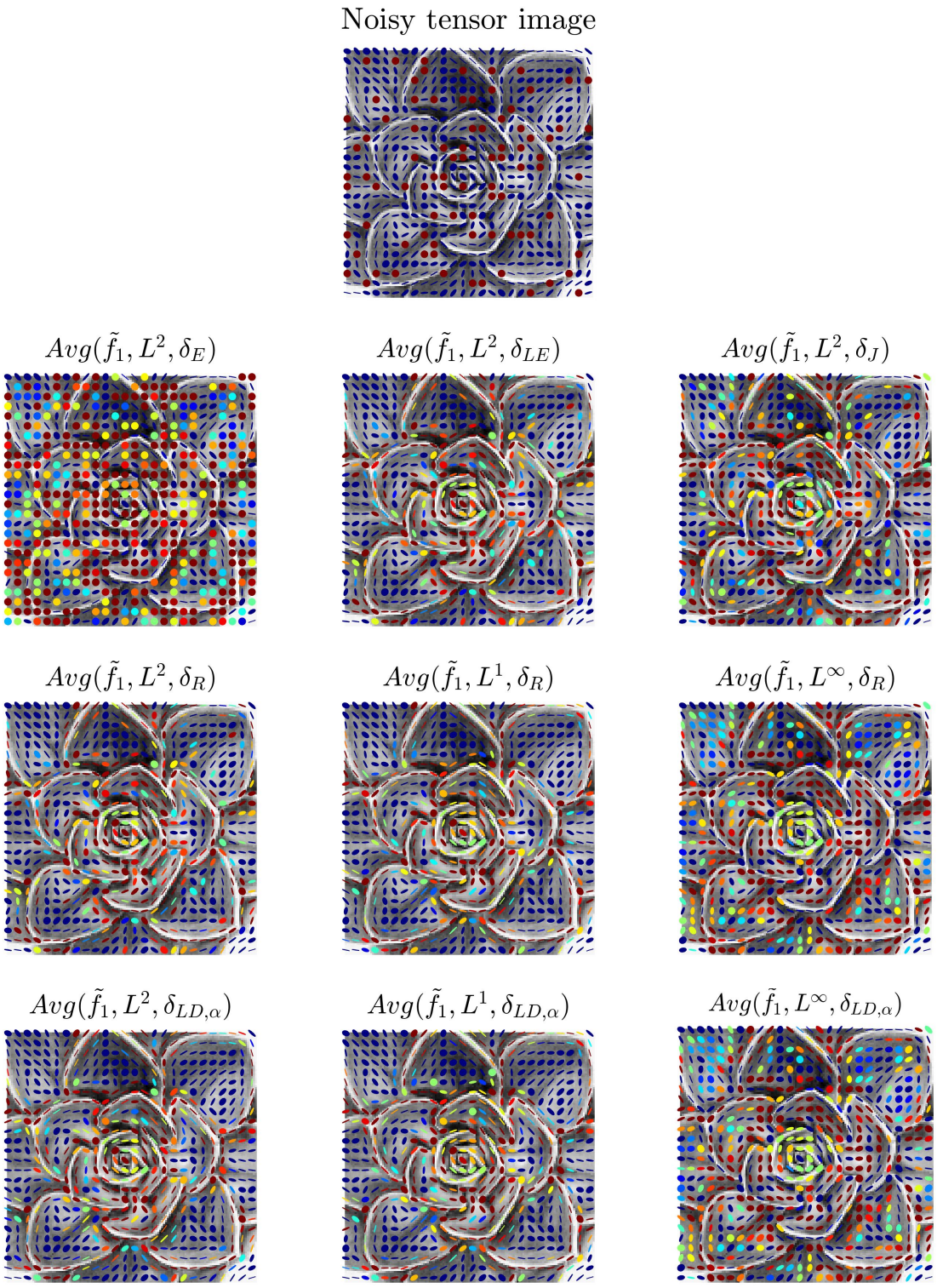


Figure 6.5: Comparison of different averaging techniques for structure tensor image denosing. First row: simulated noisy image with $Pr = 0.1$; Second-fourth row: denoised images by different averaging techniques as specified in the titles.

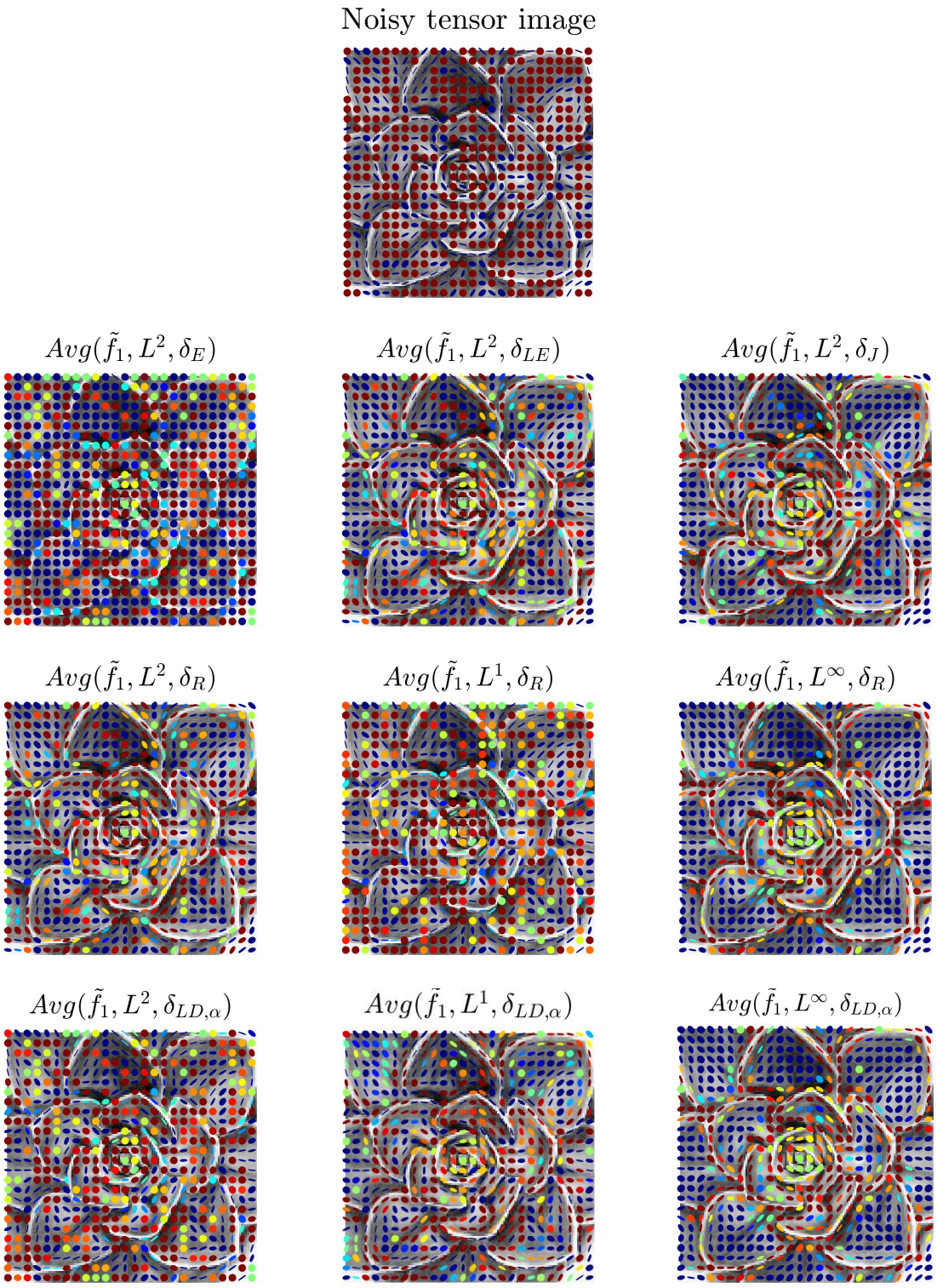


Figure 6.6: Comparison of different averaging techniques for structure tensor image denosing. First row: simulated noisy image with $Pr = 0.5$; Second-fourth row: denoised images by different averaging techniques as specified in the titles.

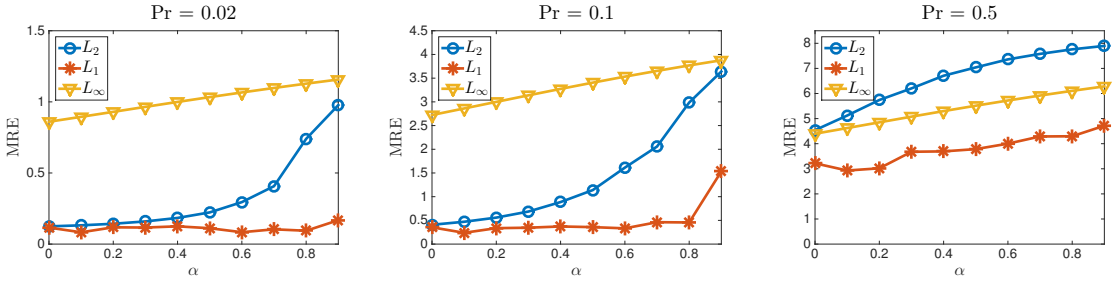


Figure 6.7: Average Mean Riemannian Error (MRE) over 10 experiments for the structure tensor image denoising based on the LogDet α-divergence with varying values of α.

Table 6.3: Performance results obtained for EEG classification using different cluster centers and distances/divergences. Each row of the table corresponds to one subject, and the last row corresponds to the average results over 12 subjects. For simplicity of notation, acc% refers to the classification accuracy in percentage and t(ms) refers to the computation time in milliseconds. Bold blue numbers indicate the best.

	Inductive		J-div.		Riemannian		α-div.		S-α-div.		Rie.-Med.		α-median		S-α-median	
Sub	acc%	t(ms)	acc%	t(ms)	acc%	t(ms)	acc%	t(ms)	acc%	t(ms)	acc%	t(ms)	acc%	t(ms)	acc%	t(ms)
1	70.31	5.08	**78.13**	0.84	73.44	19.17	59.38	5.00	**78.13**	8.22	75.00	16.37	62.50	3.90	75.00	7.61
2	78.13	5.14	78.13	0.84	**79.69**	19.61	**79.69**	5.00	78.13	8.18	**79.69**	17.72	**79.69**	4.02	**79.69**	7.65
3	85.94	5.07	92.19	0.84	85.94	18.11	**95.31**	5.12	85.94	8.12	85.94	16.31	**95.31**	4.16	85.94	7.53
4	87.50	5.05	87.50	0.84	87.50	18.82	**89.06**	5.01	**89.06**	8.20	**89.06**	15.69	**89.06**	3.87	**89.06**	7.53
5	67.19	5.08	71.88	0.84	68.75	19.87	**73.44**	5.00	70.31	7.72	68.75	19.46	70.31	3.87	68.75	7.39
6	84.38	5.08	**87.50**	0.84	85.94	20.86	**87.50**	5.01	85.94	8.15	84.38	19.03	**87.50**	3.87	84.38	7.64
7	89.58	10.91	90.62	1.22	88.54	38.90	91.67	7.55	90.62	13.17	89.58	37.29	**92.71**	6.30	89.58	12.22
8	92.19	5.11	**92.19**	0.84	**92.19**	20.00	**92.19**	5.00	**92.19**	7.78	**92.19**	19.00	**92.19**	3.87	**92.19**	7.29
9	70.31	5.07	**76.56**	0.83	70.31	19.39	75.00	4.99	75.00	8.01	71.88	18.84	73.44	3.93	71.88	7.68
10	78.91	16.83	78.13	1.59	80.47	58.02	**82.03**	9.90	80.47	18.18	79.69	54.25	80.47	8.71	79.69	15.64
11	64.06	5.02	43.75	0.83	65.63	19.92	57.81	6.02	60.94	7.98	67.19	20.93	68.75	4.60	**71.88**	8.25
12	96.88	22.67	**97.50**	1.97	96.88	80.23	95.63	12.36	96.88	23.63	96.88	74.45	95.63	11.17	96.88	20.89
Avg	80.45	8.01	81.17	1.03	81.27	29.41	81.56	6.33	81.97	10.61	81.69	27.45	**82.30**	5.19	82.08	9.78

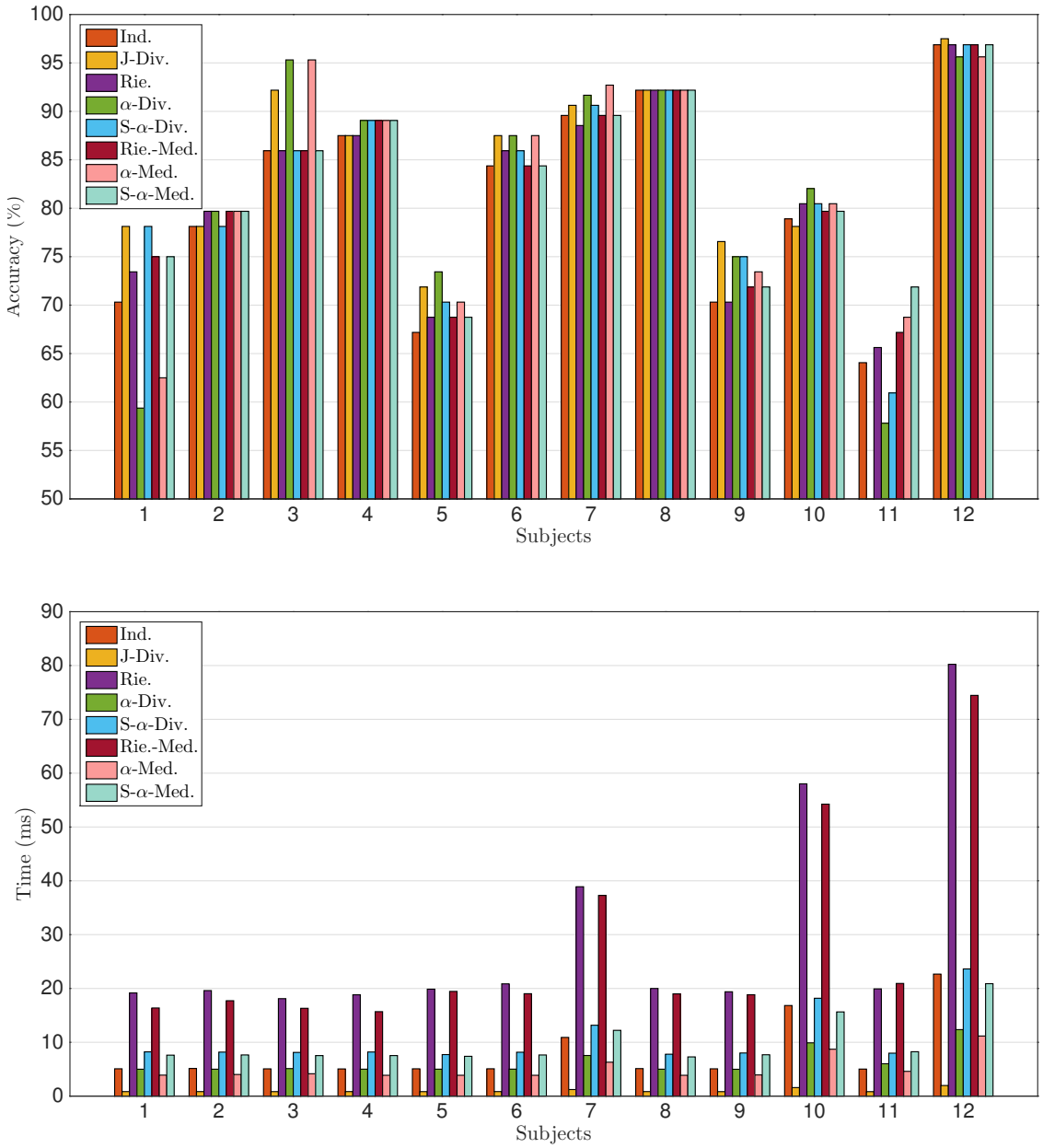


Figure 6.8: Performance results obtained for the EEG classification using different cluster centers and distances/divergences for each subject. The top plot displays the classification accuracy on the test set and the bottom plot shows the computation time required to compute the cluster center of each cluster in the training set.

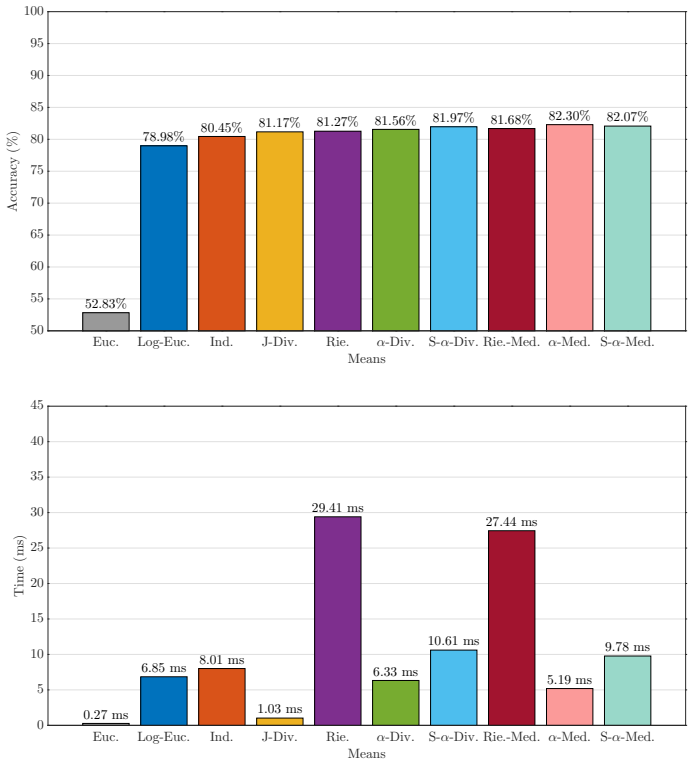


Figure 6.9: Average results over 12 subjects for the EEG classification using different cluster centers and distances/divergences.

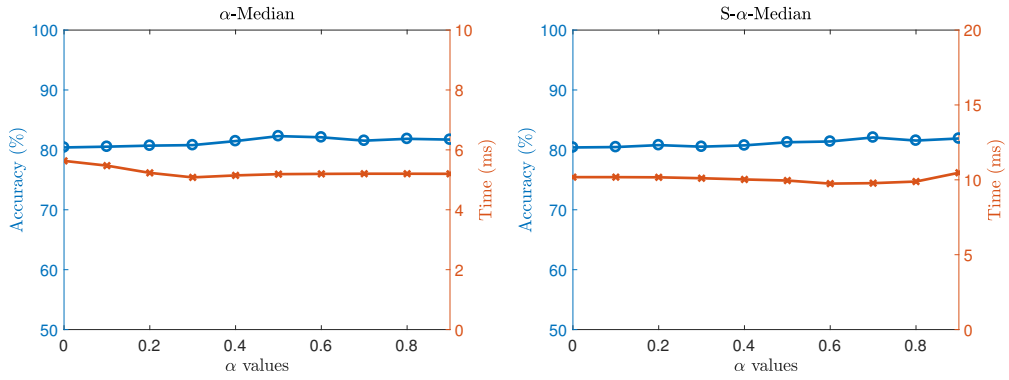


Figure 6.10: Average results over 12 subjects for the EEG classification using the LogDet α-divergence-based median and its symmetrized version with $\alpha = 0, 0.1, \ldots, 0.9$.

Table 6.4: Average classification accuracy over 12 subjects for the EEG classification using the LogDet α-divergence-based median and its symmetrized version with different values of α.

α	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
α-median (%)	80.40	80.53	80.71	80.79	81.47	**82.30**	82.10	81.54	81.84	81.71
S-α-median (%)	80.40	80.47	80.79	80.53	80.75	81.27	81.40	**82.07**	81.55	81.90

Table 6.5: Average classification accuracy over 12 subjects for the EEG classification using the arithmetic-harmonic mean combined with different distances/divergences. The Jeffrey divergence achieves the best result.

Distance/divergence	Euc.	LogEuc.	J-Div.	Rie.	α-Div.	S-α-Div.
Accuracy	57.78%	78.04%	**81.17%**	79.59%	80.95%	80.80%

Table 6.6: contingency table

		True result	
		X_i and X_j belong to the same cluster	X_i and X_j belong to different clusters
Predicted result	X_i and X_j belong to the same cluster	True Positive (TP)	False Positive (FP)
	X_i and X_j belong to different clusters	False Negative (FN)	True Negative (TN)

Table 6.7: Samples of the KTH-TIPS2 dataset for classes of wood, cotton and lettuce. Plots in the same row belong to the same class.

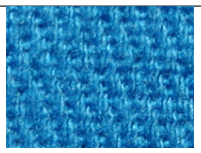
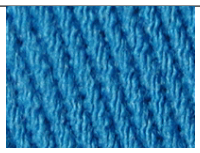
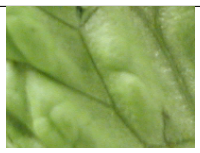
Wood			
Cotton			
Lettuce			

Table 6.8: Samples of the virus dataset for 3 different classes. Plots in the same row belong to the same class.

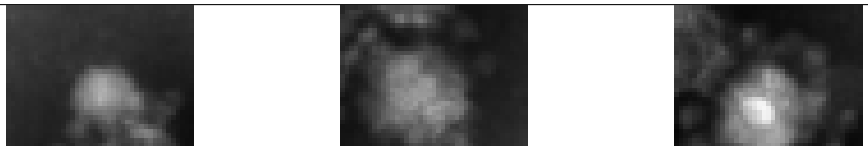
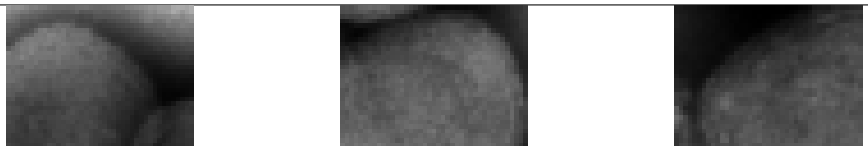
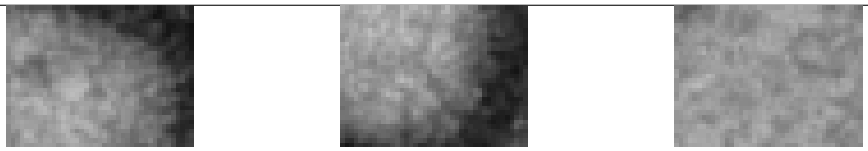
Class 1	
Class 2	
Class 3	

Table 6.9: Notation for reporting experiment results

F1	F1-score
NMI	normalized mutual information
t	time required for K-means clustering (seconds)
iter	number of iterations required for K-means clustering to converge
t/iter	time for each iteration
t_{mean}	time required to compute the cluster centers (seconds)
t_{δ}	time required to compute the distance/divergence (seconds)

Table 6.10: Comparison of K-means clustering using different cluster centers and distance/divergence on the KTH-TIPS dataset.

Means	F1 (%)	NMI (%)	t (s)	iter	t/iter (s)	t_{mean} (s)	t_{δ} (s)
Euclid	34.59	45.69	7.55	40.30	0.19	1.21	6.34
LogEuclid	40.74	52.17	478.49	**34.40**	13.87	21.18	457.31
J-div.	43.61	55.08	126.65	44.50	**2.84**	**7.69**	118.97
Inductive	44.46	55.54	454.11	49.60	9.14	42.16	411.95
Riemannian	44.68	55.68	822.56	54.80	14.97	367.90	454.66
α-div. ($\alpha = 0.9$)	45.63	**57.21**	**121.80**	40.40	3.01	53.47	**68.33**
S-α-div. ($\alpha = -0.7$)	44.83	55.91	378.01	63.60	6.06	222.73	155.28
Rie-median	44.51	55.43	517.28	36.60	14.05	216.16	301.13
α-median ($\alpha = 0.9$)	**45.75**	56.68	131.96	41.10	3.20	62.39	69.56
S-α-median ($\alpha = -0.7$)	44.76	55.74	233.65	40.90	5.71	134.43	99.23

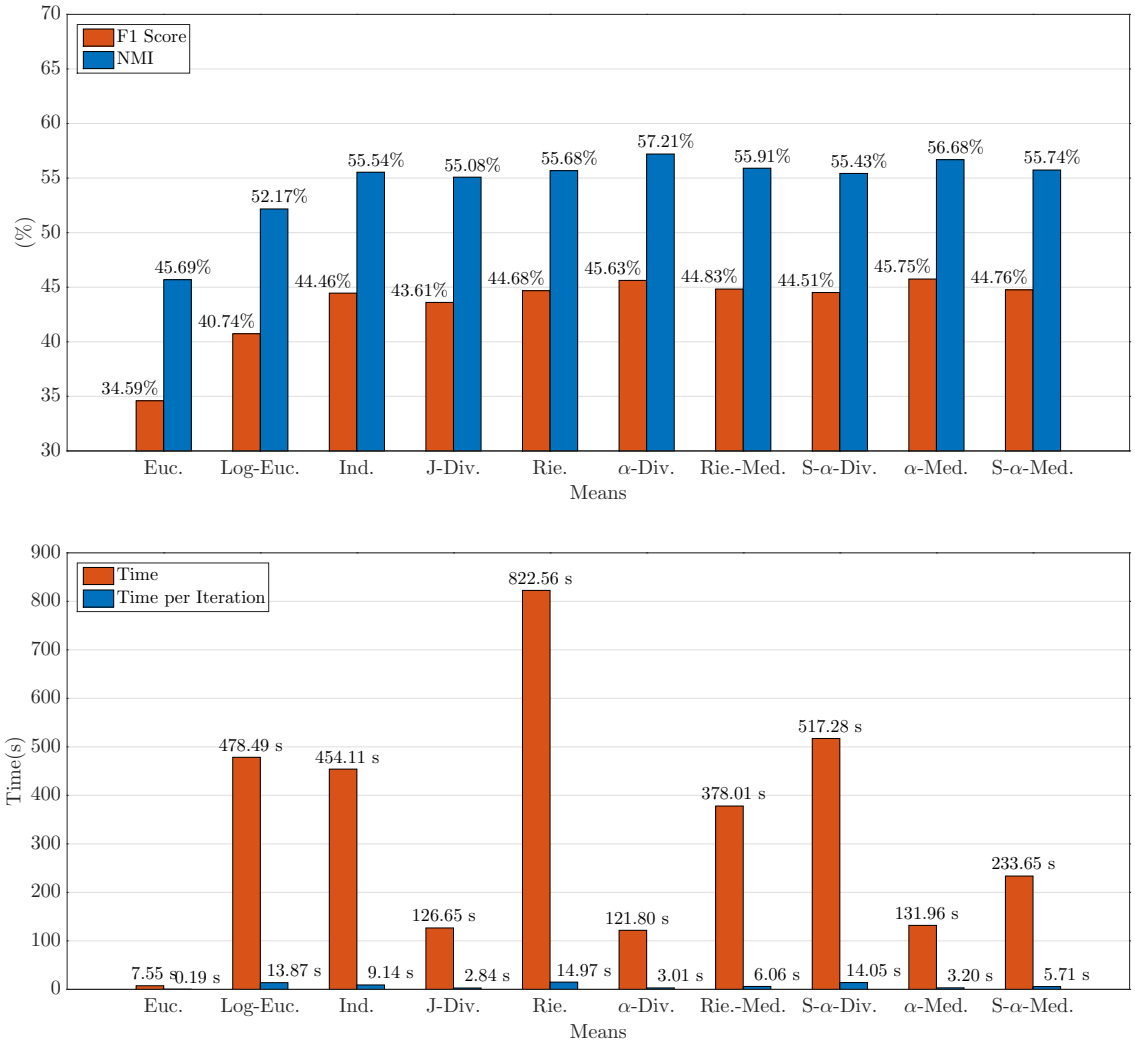


Figure 6.11: Comparison of K-means clustering using different cluster centers and distance/divergence functions on the KTH-TIPS dataset.

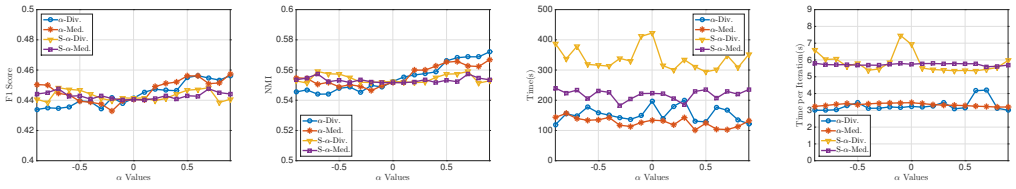


Figure 6.12: Clustering quality and computation time obtained for the LogDet α-divergence and its symmetrized version with varying values of α.

Table 6.11: Comparison of K-means clustering using different cluster centers and distance/divergence on the VIRUS dataset.

Means	F1 (%)	NMI (%)	t (s)	iter	t/iter (s)	t_{mean} (s)	t_{δ} (s)
Euclid	18.57	29.06	3.28	33	0.10	0.60	2.68
LogEuclid	24.57	36.44	274.61	29	9.34	9.24	265.36
Inductive	26.36	37.85	228.40	36	6.40	16.07	212.34
J-div.	25.86	37.19	**52.04**	**27**	1.93	**2.62**	49.42
Riemannian	26.49	37.96	332.37	33	10.00	133.72	198.65
α-div. ($\alpha = 0.1$)	**26.74**	**38.22**	57.60	30	1.89	23.22	**34.38**
S-α-div. ($\alpha = 0$)	26.56	38.03	117.87	36	3.25	56.61	61.26
Rie-median	25.72	37.57	261.35	31	8.37	77.27	184.08
α-median ($\alpha = -0.6$)	25.63	37.43	57.99	30	**1.92**	23.17	34.82
S-α-median ($\alpha = -0.5$)	26.23	37.92	121.22	38	3.22	58.79	62.44

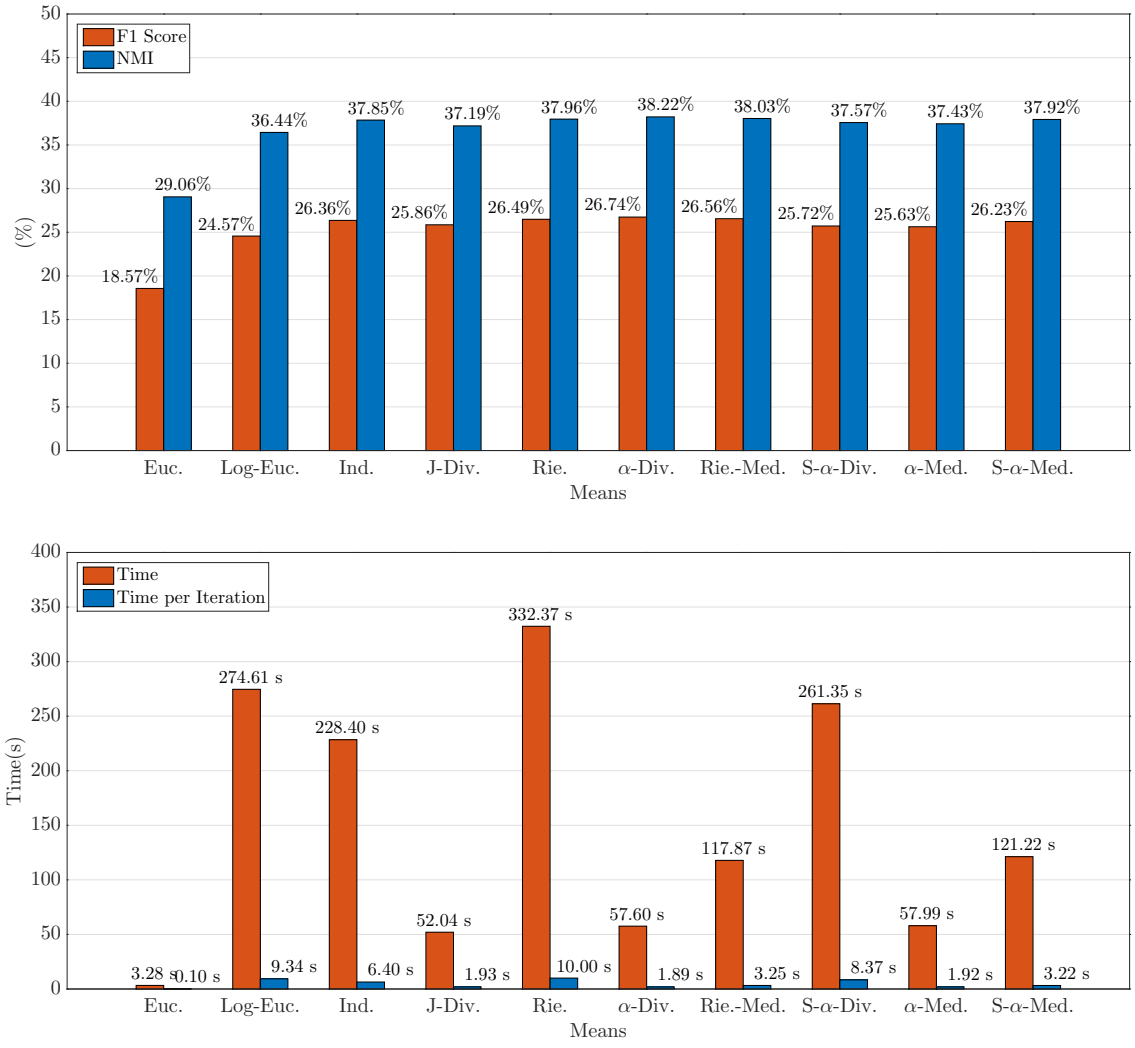


Figure 6.13: Comparison of K-means clustering using different cluster centers and distance/divergence functions on the VIRUS dataset.

CHAPTER 7

CONCLUSIONS AND FUTURE RESEARCH

This dissertation investigates different averaging techniques and similarity measures for SPD matrices. We propose to use recent developments in Riemannian optimization to develop efficient and robust algorithms to compute different central representatives of a collection of SPD matrices.

The major contributions of this dissertation are:

1. **Investigated different averaging techniques for symmetric positive definite matrices, including the computation of means, medians, and minimax centers;**

   Different definitions of means, medians and minimax centers for a collection of symmetric positive definite matrices are discussed. Different distances and divergences are considered. Various Riemannian optimization algorithms are exploited to tackle the computation of the mean, median, and minimax center.

2. **Exploited recent developments in Riemannian optimization to develop efficient and robust algorithms on the manifold of symmetric positive definite matrices;**

   Methods to produce efficient numerical representations of geometric objects that are required for Riemannian optimization methods on the manifold of symmetric positive definite matrices are provided. For the mean computation, a LRBFGS is exploited to reduce storage requirements and computation time. The modified and nonsmooth Riemannian quasi-Newton algorithms are exploited to handle nonsmooth functions in the median computation and minimax center computation problems. Theoretical and empirical suggestions on how to choose between various methods and parameters are provided. For the mean computation, this dissertation also proposes an explanation of the good performance of steepest descent methods observed in the literature; the explanation crucially relies on an upper bound on the condition number of the Riemannian Hessian of the objective function that depends on the *logarithm* of the condition number of the data matrices.

3. **Provided empirical assessments and comparisons of the performance of considered Riemannian optimization algorithms and existing stat-of-the-art algorithms;**

   Systematic numerical experiments to compare and evaluate the performance of various algorithms are conducted, taking into account the impact of data distributions, manifold dimension, initial iterate, the choice of parameters in the algorithms, etc. For different computational tasks, the preferred method is identified.

4. **Contributed a C++ toolbox to estimate means, medians, and minimax centers for a collection of SPD matrices, using different distance and divergence functions presented in this dissertation;**

   To the best of our knowledge, there is no other publicly available and comprehensive C++ toolbox for averaging SPD matrices. The Matrix Means Toolbox[1] developed by Bini et al. in [16] is written in MATLAB, and only computes the mean. As an interpreted language, MATLAB's execution efficiency is lower than compiled languages, such as C++. We resort to C++ for efficiency, and also provide empirical illustrations of the speedup using C++ implementation instead of MATLAB.

5. **Evaluated the performance of different averaging techniques in applications, including denoising, supervised classification and unsupervised clustering.**

   - Structure tensor image denoising
   - Electroencephalography (EEG) classification problem based on the Minimum Distance to Mean (MDM) classifier
   - K-means clustering on real-life data: KTH-TIPS2 dataset [66], Virus dataset [61]

There are several opportunities for future research. In the SPD median computation, it is observed in Section 5.3.1 that the nonsmooth quasi-Newton algorithms do not perform well. Modifications can be made to the nonsmooth versions such that they behave the same as the smooth versions when the cost function is smooth. That is, we consider to propose a hybrid algorithm that can handle smooth and nonsmooth scenarios. Additionally, it is observed in Figure 4.4 and Figure 4.7 that in the presence of outliers in the dataset, the limited-memory version of RBFGS outperforms the full version in terms of convergence rate and computation time. The presence of outliers tends to slow down the convergence of RBFGS, but has a smaller impact on the limited-memory version. Further study is needed to understand this phenomenon.

In addition, current averaging techniques work well on moderate-size and full-rank SPD matrices. Most implementations involve basic matrix operations, such as Cholesky factorization, eigenvalue decomposition, solving linear systems, etc. The complexity of those matrix operations grows cubically with the size of matrix, i.e., $O(n^3)$. This limits the use of averaging techniques in large-scale problems where the cubic complexity is not realistic. Using low-rank approximations of SPD matrices is a common way to reduce computational complexity. Another challenge concerns the conditioning of the data matrices. The data matrices may become so ill-conditioned that their

numerical rank is reduced. These motivate the need of extending averaging techniques from positive definite matrices to positive semidefinite matrices of fixed rank. Bonnabel et al. [19] took the first step in defining the geometric mean of two positive semidefinite matrices of fixed rank. Later [18] generalized this geometric mean to a general number of matrices. A potentially interesting avenue for future research is to explore other averaging techniques for positive semidefinite matrices of fixed rank.

BIBLIOGRAPHY

[1] P.-A. Absil and P.-Y. Gousenbourger. Differentiable piecewise-Bezier surfaces on Riemannian manifolds. Technical report, ICTEAM Institute, Universite Catholiqué de Louvain, Louvain-La-Neuve, Belgium, 2015.

[2] P.-A. Absil, R. Mahony, and R. Sepulchre. *Optimization algorithms on matrix manifolds*. Princeton University Press, 2008.

[3] B. Afsari, R. Tron, and R. Vidal. On the convergence of gradient descent for finding the Riemannian center of mass. *SIAM Journal on Control and Optimization*, 51(3):2230–2260, 2013.

[4] Bijan Afsari. Riemannian L^p center of mass: existence, uniqueness, and convexity. *Proceedings of the American Mathematical Society*, 139(2):655–673, 2011.

[5] Khaled Alyani, Marco Congedo, and Maher Moakher. Diagonality measures of Hermitian positive-definite matrices with application to the approximate joint diagonalization problem. *Linear Algebra and its Applications*, 2016.

[6] T. Ando and R. Li, C.-K.and Mathias. Geometric means. *Linear Algebra and its Applications*, 385:305–334, 2004.

[7] Jesus Angulo. Structure tensor image filtering using Riemannian L_1 and L_∞ center-of-mass. *Image Analysis & Stereology*, 33(2):95–105, 2014.

[8] Ognjen Arandjelovic, Gregory Shakhnarovich, John Fisher, Roberto Cipolla, and Trevor Darrell. Face recognition with image sets using manifold density divergence. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 581–588. IEEE, 2005.

[9] Marc Arnaudon and Frank Nielsen. On approximating the Riemannian 1-center. *Computational Geometry*, 46(1):93–104, 2013.

[10] Vincent Arsigny, Pierre Fillard, Xavier Pennec, and Nicholas Ayache. Log-Euclidean metrics for fast and simple calculus on diffusion tensors. *Magnetic resonance in medicine*, 56(2):411–421, 2006.

[11] Mihai Badoiu and Kenneth L Clarkson. Smaller core-sets for balls. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 801–802. Society for Industrial and Applied Mathematics, 2003.

[12] Alexandre Barachant, Stéphane Bonnet, Marco Congedo, and Christian Jutten. Multiclass brain–computer interface classification by Riemannian geometry. *IEEE Transactions on Biomedical Engineering*, 59(4):920–928, 2012.

[13] F. Barbaresco. Innovative tools for radar signal processing based on Cartan's geometry of SPD matrices and information geometry. In *IEEE Radar Conference*, pages 1–6, May 2008.

[14] Angelos Barmpoutis, Baba C Vemuri, Timothy M Shepherd, and John R Forder. Tensor splines for interpolation and approximation of DT-MRI with applications to segmentation of isolated rat hippocampi. *IEEE transactions on medical imaging*, 26(11):1537–1546, 2007.

[15] Rajendra Bhatia and Rajeeva L Karandikar. Monotonicity of the matrix geometric mean. *Mathematische Annalen*, 353(4):1453–1467, 2012.

[16] D. A. Bini and B. Iannazzo. Computing the Karcher mean of symmetric positive definite matrices. *Linear Algebra and its Applications*, 438(4):1700–1710, 2013.

[17] Dario Andrea Bini and Bruno Iannazzo. A note on computing matrix geometric means. *Advances in Computational Mathematics*, 35(2-4):175–192, 2011.

[18] Silvere Bonnabel, Anne Collard, and Rodolphe Sepulchre. Rank-preserving geometric means of positive semi-definite matrices. *Linear Algebra and its Applications*, 438(8):3202–3216, 2013.

[19] Silvere Bonnabel and Rodolphe Sepulchre. Riemannian metric and geometric mean for positive semidefinite matrices of fixed rank. *SIAM Journal on Matrix Analysis and Applications*, 31(3):1055–1070, 2009.

[20] Lev M Bregman. The relaxation method of finding the common point of convex sets and its application to the solution of problems in convex programming. *USSR computational mathematics and mathematical physics*, 7(3):200–217, 1967.

[21] Malek Charfi, Zeineb Chebbi, Maher Moakher, and Baba C Vemuri. Bhattacharyya median of symmetric positive-definite matrices and application to the denoising of diffusion-tensor fields. In *Biomedical Imaging (ISBI), 2013 IEEE 10th International Symposium on*, pages 1227–1230. IEEE, 2013.

[22] Zeineb Chebbi and Maher Moakher. Means of Hermitian positive-definite matrices based on the log-determinant α-divergence function. *Linear Algebra and its Applications*, 436(7):1872–1889, 2012.

[23] Guang Cheng, Hesamoddin Salehian, and Baba Vemuri. Efficient recursive algorithms for computing the mean diffusion tensor and applications to DTI segmentation. *Computer Vision–ECCV 2012*, pages 390–401, 2012.

[24] Anoop Cherian, Vassilios Morellas, and Nikolaos Papanikolopoulos. Bayesian nonparametric clustering for positive definite matrices. *IEEE transactions on pattern analysis and machine intelligence*, 38(5):862–874, 2016.

[25] Anoop Cherian, Suvrit Sra, Arindam Banerjee, and Nikolaos Papanikolopoulos. Efficient similarity search for covariance matrices via the Jensen-Bregman LogDet divergence. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2399–2406. IEEE, 2011.

[26] Anoop Cherian, Suvrit Sra, Arindam Banerjee, and Nikolaos Papanikolopoulos. Jensen-Bregman logdet divergence with application to efficient similarity search for covariance matrices. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(9):2161–2174, 2013.

[27] Andrzej Cichocki, Sergio Cruces, and Shun-ichi Amari. Log-determinant divergences revisited: Alpha-beta and gamma log-det divergences. *Entropy*, 17(5):2988–3034, 2015.

[28] Frank H Clarke. *Optimization and nonsmooth analysis*, volume 5. Siam, 1990.

[29] Thomas M Cover and Joy A Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

[30] Inderjit S Dhillon and Joel A Tropp. Matrix nearness problems with Bregman divergences. *SIAM Journal on Matrix Analysis and Applications*, 29(4):1120–1146, 2007.

[31] Daniela di Serafinoa, Valeria Ruggierob, Gerardo Toraldoc, and Luca Zannid. On the steplength selection in gradient methods for unconstrained optimization. 2017.

[32] Ian L Dryden, Alexey Koloydenko, and Diwei Zhou. Non-Euclidean statistics for covariance matrices, with applications to diffusion tensor imaging. *The Annals of Applied Statistics*, pages 1102–1123, 2009.

[33] José-Jesús Fernández and Sam Li. An improved algorithm for anisotropic nonlinear diffusion for denoising cryo-tomograms. *Journal of structural biology*, 144(1-2):152–161, 2003.

[34] P. T. Fletcher and S. Joshi. Riemannian geometry for the statistical analysis of diffusion tensor data. *Signal Processing*, 87(2):250–262, 2007.

[35] P. T. Fletcher, C. Lu, S. M. Pizer, and S. Joshi. Principal geodesic analysis on symmetric spaces: statistics of diffusion tensors. *Computer Vision and Mathematical Methods in Medical and Biomedical Image Analysis*, 3117:87–98, 2004.

[36] P Thomas Fletcher, Suresh Venkatasubramanian, and Sarang Joshi. Robust statistics on Riemannian manifolds via the geometric median. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

[37] P Thomas Fletcher, Suresh Venkatasubramanian, and Sarang Joshi. The geometric median on Riemannian manifolds with application to robust atlas estimation. *NeuroImage*, 45(1):S143–S152, 2009.

[38] Wolfgang Förstner and Eberhard Gülch. A fast operator for detection and precise location of distinct points, corners and centres of circular features. In *Proc. ISPRS intercommission conference on fast processing of photogrammetric data*, pages 281–305. Interlaken, 1987.

[39] Giacomo Frassoldati, Luca Zanni, and Gaetano Zanghirati. New adaptive stepsize selections in gradient methods. *Journal of industrial and management optimization*, 4(2):299, 2008.

[40] Gene H Golub and Charles F Van Loan. *Matrix computations*, volume 3. JHU Press, 2012.

[41] Martin Gröger, Wolfgang Sepp, Tobias Ortmaier, and Gerd Hirzinger. Reconstruction of image structure in presence of specular reflections. In *Joint Pattern Recognition Symposium*, pages 53–60. Springer, 2001.

[42] Mehrtash Harandi, Mina Basirat, and Brian C Lovell. Coordinate coding on the Riemannian manifold of symmetric positive-definite matrices for image classification. In *Riemannian Computing in Computer Vision*, pages 345–361. Springer, 2016.

[43] Mehrtash Harandi, Mathieu Salzmann, and Fatih Porikli. Bregman divergences for infinite dimensional covariance matrices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1003–1010, 2014.

[44] Nicholas J Higham. *Functions of matrices: theory and computation*. SIAM, 2008.

[45] Seyedehsomayeh Hosseini, Wen Huang, and Rohollah Yousefpour. Line search algorithms for locally Lipschitz functions on Riemannian manifolds. *SIAM Journal on Optimization*, 28(1):596–619, 2018.

[46] Wen Huang. *Optimization algorithms on Riemannian manifolds with applications*. PhD thesis, Department of Mathematics, Florida State University, 2012.

[47] Wen Huang, P-A Absil, and Kyle A Gallivan. A Riemannian symmetric rank-one trust-region method. *Mathematical Programming*, 150(2):179–216, 2015.

[48] Wen Huang, P-A Absil, and Kyle A Gallivan. Intrinsic representation of tangent vectors and vector transports on matrix manifolds. *Numerische Mathematik*, pages 1–21, 2016.

[49] Wen Huang, P-A Absil, and Kyle A Gallivan. A Riemannian BFGS method for nonconvex optimization problems. In *Numerical Mathematics and Advanced Applications ENUMATH 2015*, pages 627–634. Springer, 2016.

[50] Wen Huang, PA Absil, KA Gallivan, and Paul Hand. ROPTLIB: an object-oriented C++ library for optimization on Riemannian manifolds. Technical report, Technical Report FSU16-14, Florida State University, 2016.

[51] Wen Huang, Kyle A Gallivan, and P-A Absil. A Broyden class of quasi-Newton methods for Riemannian optimization. *SIAM Journal on Optimization*, 25(3):1660–1685, 2015.

[52] Bruno Iannazzo and Margherita Porcelli. The Riemannian Barzilai–Borwein method with non-monotone line search and the matrix geometric mean computation. *IMA Journal of Numerical Analysis*, 38(1):495–517, 2017.

[53] B. Jeuris, R. Vandebril, and B. Vandereycken. A survey and comparison of contemporary algorithms for computing the matrix geometric mean. *Electronic Transactions on Numerical Analysis*, 39:379–402, 2012.

[54] Ben Jeuris and Raf Vandebril. Geometric mean algorithms based on harmonic and arithmetic iterations. In *Geometric Science of Information*, pages 785–793. Springer, 2013.

[55] Emmanuel K Kalunga, Sylvain Chevallier, Quentin Barthélemy, Karim Djouani, Yskandar Hamam, and Eric Monacelli. From Euclidean to Riemannian means: Information geometry for SSVEP classification. In *International Conference on Networked Geometric Science of Information*, pages 595–604. Springer, 2015.

[56] H. Karcher. Riemannian center of mass and mollifier smoothing. *Communications on Pure and Applied Mathematics*, 1977.

[57] Fumio Kubo and Tsuyoshi Ando. Means of positive linear operators. *Mathematische Annalen*, 246(3):205–224, 1980.

[58] Gerald Kuhne, Joachim Weickert, Oliver Schuster, and Stephan Richter. A tensor-driven active contour model for moving object segmentation. In *Image Processing, 2001. Proceedings. 2001 International Conference on*, volume 2, pages 73–76. IEEE, 2001.

[59] Brian Kulis, Mátyás Sustik, and Inderjit Dhillon. Learning low-rank kernel matrices. In *Proceedings of the 23rd international conference on Machine learning*, pages 505–512. ACM, 2006.

[60] Brian Kulis, Mátyás A Sustik, and Inderjit S Dhillon. Low-rank kernel learning with Bregman matrix divergences. *Journal of Machine Learning Research*, 10(Feb):341–376, 2009.

[61] Gustaf Kylberg, Mats Uppström, K-O Hedlund, Gunilla Borgefors, and I-M Sintorn. Segmentation of virus particle candidates in transmission electron microscopy images. *Journal of microscopy*, 245(2):140–147, 2012.

[62] J. Lapuyade-Lahorgue and F. Barbaresco. Radar detection using Siegel distance between autoregressive processes, application to HF and X-band radar. In *IEEE Radar Conference*, pages 1–6, May 2008.

[63] J. Lawson and Y. Lim. Monotonic properties of the least squares mean. *Mathematische Annalen*, 351(2):267–279, 2011.

[64] Stuart Lloyd. Least squares quantization in PCM. *IEEE transactions on information theory*, 28(2):129–137, 1982.

[65] Hendrik P Lopuhaa and Peter J Rousseeuw. Breakdown points of affine equivariant estimators of multivariate location and covariance matrices. *The Annals of Statistics*, pages 229–248, 1991.

[66] P Mallikarjuna, M Fritz, A Tavakoli Targhi, E Hayman, B Caputo, and JO Eklundh. The KTH-TIPS and KTH-TIPS2 databases, 2006.

[67] Estelle M Massart and Sylvain Chevallier. Inductive means and sequences applied to online classication of EEG. Technical report, ICTEAM Institute, Universite Catholiqué de Louvain, 2017.

[68] Estelle M Massart, Julien M Hendrickx, and P-A Absil. Matrix geometric means based on shuffled inductive sequences. *Linear Algebra and its Applications*, 542:334–359, 2018.

[69] Marina Meila. Comparing clusterings by the variation of information. In *Colt*, volume 3, pages 173–187. Springer, 2003.

[70] Rudolf Mester. A new view at differential and tensor-based motion estimation schemes. In *Joint Pattern Recognition Symposium*, pages 321–329. Springer, 2003.

[71] M. Moakher. On the averaging of symmetric positive-definite tensors. *Journal of Elasticity*, 82(3):273–296, 2006.

[72] Maher Moakher and Philipp G Batchelor. Symmetric positive-definite matrices: from geometry to applications and visualization. In *Visualization and Processing of Tensor Fields*, pages 285–298. Springer, 2006.

[73] Yurii Nesterov. Introductory lectures on convex programming volume I: Basic course. *Lecture notes*, 1998.

[74] Frank Nielsen, Meizhu Liu, Xiaojing Ye, and Baba C Vemuri. Jensen divergence based SPD matrix means and applications. In *21st International Conference on Pattern Recognition (ICPR)*, pages 2841–2844. IEEE, 2012.

[75] Frank Nielsen and Richard Nock. Total Jensen divergences: definition, properties and clustering. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, pages 2016–2020. IEEE, 2015.

[76] Michael A Nielsen and Isaac Chuang. Quantum computation and quantum information, 2002.

[77] Lawrence M Ostresh Jr. On the convergence of a class of iterative methods for solving the Weber location problem. *Operations Research*, 26(4):597–609, 1978.

[78] X. Pennec, P. Fillard, and N. Ayache. A Riemannian framework for tensor computing. *International Journal of Computer Vision*, 66(1):41–66, 2006.

[79] Robert Pless and David Jurgens. Road extraction from motion cues in aerial video. In *Proceedings of the 12th annual ACM international workshop on Geographic information systems*, pages 31–38. ACM, 2004.

[80] Tamas Rapcsak. Geodesic convexity in nonlinear optimization. *Journal of Optimization Theory and Applications*, 69(1):169–183, 1991.

[81] Y. Rathi, A. Tannenbaum, and O. Michailovich. Segmenting images on the tensor manifold. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, June 2007.

[82] Q. Rentmeesters. *Algorithms for data fitting on some common homogeneous spaces.* PhD thesis, Universite catholiqué de Louvain, 2013.

[83] Q. Rentmeesters and P.-A. Absil. Algorithm comparison for Karcher mean computation of rotation matrices and diffusion tensors. In *19th European Signal Processing Conference*, pages 2229–2233, Aug 2011.

[84] Suvrit Sra. Positive definite matrices and the S-divergence. *arXiv preprint arXiv:1110.1773*, 2011.

[85] Suvrit Sra. A new metric on the manifold of kernel matrices with application to matrix geometric means. In *Advances in neural information processing systems*, pages 144–152, 2012.

[86] Panagiotis Stanitsas, Anoop Cherian, Vassilios Morellas, and Nikolaos Papanikolopoulos. Clustering positive definite matrices by learning information divergences. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1304–1312, 2017.

[87] A Strehl and J Chosh. Knowledge reuse framework for combining multiple partitions. *Journal of Machine learning Research*, 33(3):583–617.

[88] Koji Tsuda, Gunnar Rätsch, and Manfred K Warmuth. Matrix exponentiated gradient updates for on-line learning and Bregman projection. *Journal of Machine Learning Research*, 6(Jun):995–1018, 2005.

[89] Baba C Vemuri, Meizhu Liu, Shun-Ichi Amari, and Frank Nielsen. Total Bregman divergence and its applications to DTI analysis. *IEEE Transactions on medical imaging*, 30(2):475–483, 2011.

[90] Silke Wagner and Dorothea Wagner. *Comparing clusterings: an overview*. Universität Karlsruhe, Fakultät für Informatik Karlsruhe, 2007.

[91] Zhizhou Wang and Baba C Vemuri. An affine invariant tensor dissimilarity measure and its applications to tensor-valued image segmentation. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 1, pages I–I. IEEE, 2004.

[92] Endre Weiszfeld. Sur le point pour lequel la somme des distances de n points donnés est minimum. *Tohoku Mathematical Journal, First Series*, 43:355–386, 1937.

[93] S. J. Wright and J. Nocedal. *Numerical optimization*. Springer New York, 2 edition, 2006.

[94] Florian Yger, Maxime Berar, and Fabien Lotte. Riemannian approaches in brain-computer interfaces: a review. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 25(10):1753–1762, 2017.

[95] Florian Yger, Fabien Lotte, and Masashi Sugiyama. Averaging covariance matrices for EEG signal classification based on the CSP: An empirical study. In *Signal Processing Conference (EUSIPCO), 2015 23rd European*, pages 2721–2725. IEEE, 2015.

[96] Xinru Yuan, Wen Huang, P.-A. Absil, and Kyle A. Gallivan. A Riemannian limited-memory BFGS algorithm for computing the matrix geometric mean. *Procedia Computer Science*, 80:2147–2157, 2016.

[97] Alan L Yuille and Anand Rangarajan. The concave-convex procedure. *Neural computation*, 15(4):915–936, 2003.

[98] Hongyi Zhang and Suvrit Sra. First-order methods for geodesically convex optimization. *arXiv preprint arXiv:1602.06053*, 2016.

[99] Jun Zhang. Divergence function, duality, and convex analysis. *Neural Computation*, 16(1):159–195, 2004.

BIOGRAPHICAL SKETCH

Xinru Yuan, daughter of Junhua Li and Yin Yuan, was born on April 3rd, 1989 in Kunyang, Yunnan province of P.R. China. She finished her Bachelor degree in mathematics in 2011 at the University of Science and Technology of China. She enrolled in the Ph.D. program at Florida State University on August 2012 and worked with Prof. Kyle A. Gallivan and Prof. Pierre-Antoine Absil.

Xinru's research interests include optimization algorithms on Riemannian manifolds and different averaging techniques for symmetric positive definite matrices.