MATHICSE Technical Report
Nr. 14.2012
March 2012

A survey and comparison of contemporary algorithms for computing the matrix geometric mean

Ben Jeuris, Raf Vandebril, Bart Vandereycken

A SURVEY AND COMPARISON OF CONTEMPORARY ALGORITHMS FOR COMPUTING THE MATRIX GEOMETRIC MEAN

BEN JEURIS*, RAF VANDEBRIL*, AND BART VANDEREYCKEN†

Abstract. In this paper, we present a survey of various existing algorithms for computing the matrix geometric mean and derive new second-order optimization algorithms to compute the Karcher mean. These new algorithms are constructed using the standard definition of the Riemannian Hessian. The survey includes the ALM-list of desired properties for a geometric mean, the analytical expression for the mean of two matrices, algorithms based on centroid computation in Euclidean (flat) space, and Riemannian optimization techniques to compute the Karcher mean (preceded by a short introduction into differential geometry). A change of metric is considered in the optimization techniques to reduce the complexity of the structures used in these algorithms. Numerical experiments are presented to compare the existing and the newly developed algorithms. We conclude that currently the first-order optimization algorithms are best suited for this optimization problem as the size and/or number of the matrices increases.

Key words. Matrix geometric mean, Positive definite matrices, Karcher mean, Riemannian optimization

AMS subject classifications. 15A24, 53B21, 65K10

1. Introduction.

A mean is, in general, simply a center size subjected to certain generic properties such as idempotency (the mean of $(A, \ldots, A)$ equals A), invariance under a permutation of the elements and homogeneity (the mean of $(\lambda A_1, \ldots, \lambda A_k)$ equals λ times the mean of $(A_1, \ldots, A_k)$). However, these generic properties alone do not uniquely define a mean, so there can be many different types of means. In this paper we discuss the *geometric mean*, which for positive real numbers $(a_1, \ldots, a_k)$ is defined as

$$\mathbf{G}(a_1, \ldots, a_k) = (a_1 \cdots a_k)^{\frac{1}{k}}. \tag{1.1}$$

When conveying this definition to the set of symmetric positive definite matrices $\mathbb{S}_+^n$, we see that the formula above can not be readily extended to matrices due to their non-commutativity. However, a list of desired properties for the general geometric mean can be derived from this scalar expression.

These properties (listed in section 2) have proven to be useful in various applications, e.g., radar technology [4], medical imaging [14], mechanics [20] and image processing [26]. All these areas display situations in which the information about the current system is being represented in a collection of positive definite matrices. In order to perform calculations on these matrices, such as averaging and interpolation, we need algorithms that preserve the structure of the data, being positive definiteness, which is one of the useful properties of the geometric mean. Another property of this mean provides advantages in the area of elasticity calculations of structures [20]. In these calculations, both a positive definite elasticity matrix and its inverse, the compliance matrix, are used. Hence, given a collection of these elasticity matrices and a collection consisting of the corresponding compliance matrices, then the geometric means of both matrix collections will again be each others inverse (as stated in property 8 in section 2).

Thanks to the wide range of practical and theoretical applications, matrix means have recieved a lot of attention from present-day scientists. Monoticity, for example, is a property only recently proven for the Karcher mean [6, 17, 20], which is a specific instance of a geometric mean for positive definite matrices. A consequence of the diversity of application

*Department of Computer Science, Katholieke Universiteit Leuven, Celestijnenlaan 200A, 3001 Heverlee, Belgium (ben.jeuris@cs.kuleuven.be;raf.vandebril@cs.kuleuven.be).

†Chair of ANCHP, Mathematics Section, École Polytechnique Fédérale de Lausanne, 1015 Lausanne, Switzerland (bart.vandereycken@epfl.ch).

areas is the wide variety of approaches to defining and computing the mean. Some constructions are based on intuitive interpretations of the geometric mean (section 3), while others prefer to think of it as an optimization problem (section 4). In this last approach, Riemannian optimization, which generalizes classical optimization techniques, is a popular concept.

The main contribution of this paper is to present a survey of algorithms for computing the matrix geometric mean. We recall the theoretical foundation for the analytically known mean of two matrices, the interpretations of the algorithms based on intuitive approaches and a basic framework needed to understand the methods based on Riemannian optimization. We also introduce an unprecedented, explicit expression for the Riemannian Hessian and consider the use of a different inner product on the manifold of positive definite matrices $\mathbb{S}_+^n$, which leads to simpler optimization algorithms. Finally, a first-time application of the Riemannian BFGS method to the optimization problem is presented. Numerical experiments are performed to compare all these techniques.

The organization of this paper is as follows: we start by listing the desired properties of the geometric mean and the resulting unique definition in case of two matrices in section 2. Next, in section 3, we discuss some intuitively appealing algorithms based on planar approaches: the ALM, NBMP and the CHEAP mean. However, these appealing interpretations will not lead to very efficient numerical algorithms. Finally, in section 4, we examine Riemannian optimization algorithms for the Karcher mean, which is defined as the minimizer over all positive definite matrices of the sum of squared (intrinsic) distances to all matrices in the mean. The algorithms are adapted versions of the steepest descent, conjugate gradient, trust region and BFGS methods, generalized towards manifolds. Throughout the paper we compare the performance of the algorithms discussed.

2. The geometric mean of two matrices. The scalar geometric mean (1.1) can not be readily extended to positive definite matrices because the matrix product is not commutative. Indeed, $(A_1 \cdots A_k)^{1/k}$ is not invariant under permutation, which is one of the most basic properties of means. Recall that this expression, with $A_1, \ldots, A_k \in \mathbb{S}_+^n$, is well-defined by using the convention that a matrix function f of a symmetric matrix A is determined by

$$f(A) = V f(D) V^T,$$

where $A = VDV^T$ is an eigenvalue decomposition and $f(D)$ denotes the elementwise application of f to the diagonal of D [16]. Hence a list of desired properties has been composed instead, often referred to as the ALM-list [3, 7]. Because of the importance of these properties, we summarize them here, using the partial ordering of symmetric matrices: a positive semidefinite matrix A is denoted by $A \geq 0$. Similarly, $B \geq C$ is a simplified notation for $B - C \geq 0$. The same approach is used for positive definiteness with the strict inequality. The ALM-list, using positive definite matrices $A_1, \ldots, A_k$, where we denote the *geometric mean* by $\mathbf{G}(A_1, \ldots, A_k)$, is

1. Consistency: if $A_1, \ldots, A_k$ commute, then $\mathbf{G}(A_1, \ldots, A_k) = (A_1 \ldots A_k)^{\frac{1}{k}}$.
2. Joint homogeneity:
$$\mathbf{G}(\alpha_1 A_1, \ldots, \alpha_k A_k) = (\alpha_1 \ldots \alpha_k)^{\frac{1}{k}} \mathbf{G}(A_1, \ldots, A_k), \qquad \alpha_1, \ldots, \alpha_k > 0.$$
3. Invariance under permutation: $\mathbf{G}(A_{\pi(1)}, \ldots, A_{\pi(k)}) = \mathbf{G}(A_1, \ldots, A_k)$ with π a permutation of $(1, \ldots, k)$.
4. Monotonicity: if $A_i \geq B_i$, for all i, then $\mathbf{G}(A_1, \ldots, A_k) \geq \mathbf{G}(B_1, \ldots, B_k)$.
5. Continuity from above: if for all fixed i, $A_i^{(j)}$ is a monotonously decreasing sequence of matrices converging to A_i^* for $j \to \infty$, then $\mathbf{G}(A_1^{(j)}, \ldots, A_k^{(j)})$ converges to $\mathbf{G}(A_1^*, \ldots, A_k^*)$.

6. Congruence invariance: for all invertible matrices $S \in \mathbb{R}^{n\times n}$,

$$\mathbf{G}(S^T A_1 S, \ldots, S^T A_k S) = S^T \mathbf{G}(A_1, \ldots, A_k) S.$$

7. Joint concavity:

$$\begin{aligned}&\mathbf{G}(\lambda A_1 + (1-\lambda)B_1, \ldots, \lambda A_k + (1-\lambda)B_k)\\ &\geq \lambda \mathbf{G}(A_1, \ldots, A_k) + (1-\lambda)\mathbf{G}(B_1, \ldots, B_k), \qquad 0 < \lambda < 1.\end{aligned}$$

8. Invariance under inversion: $\mathbf{G}(A_1, \ldots, A_k) = \left(\mathbf{G}(A_1^{-1}, \ldots, A_k^{-1})\right)^{-1}$.
9. Determinant equality: $\det \mathbf{G}(A_1, \ldots, A_k) = (\det A_1 \ldots \det A_k)^{\frac{1}{k}}$.
10. Arithmetic-Geometric-Harmonic inequality:

$$\frac{1}{k}\sum_{i=1}^{k} A_i \geq \mathbf{G}(A_1, \ldots, A_k) \geq \left(\frac{1}{k}\sum_{i=1}^{k} A_i^{-1}\right)^{-1}.$$

Unfortunately, these properties do not result in a unique definition for the geometric mean of a general number of matrices. For the case of two matrices, however, the geometric mean is uniquely defined from properties 1 to 10 and given by the following expressions [5]

$$\mathbf{G}(A, B) = A(A^{-1}B)^{1/2} = A^{1/2}(A^{-1/2}BA^{-1/2})^{1/2}A^{1/2}. \tag{2.1}$$

Considering the manifold of positive definite matrices $\mathbb{S}_+^n$, we can find another intuitively attractive interpretation for this result. First we note that the intrinsic distance between $A, B \in \mathbb{S}_+^n$ (see section 4.1) is given by

$$\delta(A, B) = ||\log(A^{-1/2}BA^{-1/2})||_F, \tag{2.2}$$

with $||\cdot||_F$ the Frobenius norm. Using this distance measure, we can determine the geodesic between A and B [5], i. e., the curve of shortest distance on the manifold between A and B, as

$$\begin{aligned}\gamma(t) &= A(A^{-1}B)^t = A^{1/2}(A^{-1/2}BA^{-1/2})^t A^{1/2}\\ &= A\#_t B, \quad t \in [0, 1].\end{aligned} \tag{2.3}$$

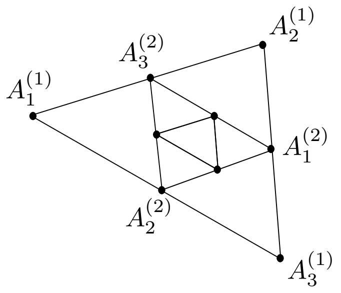
This shows that the geometric mean is exactly the midpoint on the geodesic (the notation in the last term will be used further in the text):

$$\mathbf{G}(A, B) = \gamma(1/2) = A\#_{1/2}B.$$

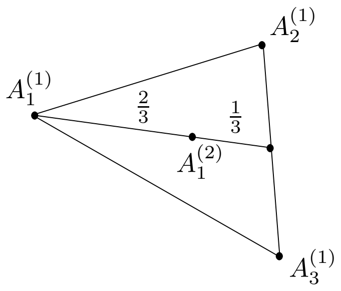
The subscript in the last term is often dropped when $t = 1/2$.

3. Geometric means based on planar approaches.

While the properties in the ALM-list result in an explicit, unique definition for calculating the geometric mean of two matrices, this is not the case when dealing with more matrices. Considering the simplified case of a space with planar Euclidean geometry, the geometric mean of three matrices is the centroid of the triangle they form. There are various intuitively appealing techniques to determine this centroid that have been generalized to the non-planar (non-Euclidean) geometry of $\mathbb{S}_+^n$ [3, 7, 9, 21, 24]. This generalization causes the need for the exact formulas for the centroid (in case of the NBMP and CHEAP mean) to be iterated. Throughout the different algorithms, we consistently notice a trade-off between the speed of convergence and the number of properties in the ALM-list the algorithms satisfy.



(a) The ALM mean.



(b) The NBMP mean.

FIGURE 3.1. *Simplified representations of the algorithms for three matrices in a flat geometric space.*

In this section, we discuss the geometric interpretation of the ALM [3], NBMP [7, 21] and CHEAP [9] mean and end with a comparison of these algorithms. In [24], the construction of new geometric means by combining existing ones is handled. Since it has been shown that there can be no comparable improvement for more than four matrices, we omit this approach in this survey.

3.1. ALM mean. The ALM mean [3] is a geometric mean which, as the name implies, satisfies the desired properties enumerated in the ALM-list. When taking the ALM mean of k matrices, recursion is used to define the iterations in which we replace $\left(A_1^{(j)}, ..., A_k^{(j)}\right)$ by

$$\left(A_1^{(j+1)}, ..., A_k^{(j+1)}\right) = \left(G_{ALM}((A_i^{(j)})_{i\neq 1}), ..., G_{ALM}((A_i^{(j)})_{i\neq k})\right),$$

where G_{ALM} denotes the recursively defined ALM mean of $k-1$ matrices with the known geometric mean of two matrices (2.1) as its base. In [3], all terms in these iterations are proven to converge towards the same limit and in figure 3.1a, a planar simplification of this algorithm for three matrices is depicted.

3.2. NBMP mean. The NBMP mean [7, 21], just as the ALM mean, satisfies all properties in the ALM-list. To compute the NBMP mean of k matrices, we use recursion to define the iterations in which we replace $\left(A_1^{(j)}, ..., A_k^{(j)}\right)$ by $\left(A_1^{(j+1)}, ..., A_k^{(j+1)}\right)$, which equals

$$\left(A_1^{(j)} \#_{\frac{k-1}{k}} G_{NBMP}((A_i^{(j)})_{i\neq 1}), ..., A_k^{(j)} \#_{\frac{k-1}{k}} G_{NBMP}((A_i^{(j)})_{i\neq k})\right),$$

where G_{NBMP} denotes the recursively defined NBMP mean of $k-1$ matrices, with the geometric mean of two matrices (2.1) as its base. The notation from (2.3) was used to denote the point on the geodesic representing the weigthed mean of the terms involved. In [7], all terms in these iterations are again proven to converge towards the same limit and in figure 3.1b we show a simplified representation of the algorithms for three matrices.

3.3. General class. We have encountered two means, both satisfying all properties in the ALM-list, but yielding different results, as shown in the next example.

EXAMPLE 3.1. If we consider the matrices

$$\begin{bmatrix} 25 & 4 \\ 4 & 1 \end{bmatrix}, \begin{bmatrix} 20 & 1 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 1 & 20 \end{bmatrix},$$

the results for the ALM and NBMP algorithm are respectively

$$\begin{bmatrix} 7.6943 & 0.9919 \\ 0.9919 & 2.0528 \end{bmatrix} \text{ and } \begin{bmatrix} 7.7139 & 0.9719 \\ 0.9719 & 2.0425 \end{bmatrix},$$

which clearly shows that the results differ.

In fact, in [7], it is showed that the ALM and NBMP mean are two instances of an entire class of means, all satisfying the required properties but with possible different results. For k matrices this *general mean* $G_{s_1,\ldots,s_{k-1}}$ depends on $k-1$ parameters $(s_1,\ldots,s_{k-1})$ and again recursion is used to define the iterations, in which we replace $\left(A_1^{(j)}, \ldots, A_k^{(j)}\right)$ by $\left(A_1^{(j+1)}, \ldots, A_k^{(j+1)}\right)$, defined as

$$\left(A_1^{(j)} \#_{s_1} G_{s_2,\ldots,s_{k-1}}((A_i^{(j)})_{i\neq 1}), \ldots, A_k^{(j)} \#_{s_1} G_{s_2,\ldots,s_{k-1}}((A_i^{(j)})_{i\neq k})\right).$$

For the ALM and NBMP mean, these parameters become respectively $(1,1,\ldots,1,1/2)$ and $((k-1)/k,(k-2)/(k-1),\ldots,1/2)$. This class illustrates that for a general number of matrices the geometric mean is not uniquely defined, not even starting from the ten desired properties. In section 4, we investigate the Karcher mean, which also satisfies all properties but has a more appealing analogy with the arithmetic mean.

3.4. CHEAP mean. The CHEAP mean [9], unlike the previous algorithms, is no longer recursively defined. It also no longer satisfies all properties present in the ALM-list, but as we will notice later, this will be compensated by its very cheap computational cost. The underlying idea is again computing the centroid of a triangle (with vertices A, B, and C, see figure 3.2) by the formula

$$A + \frac{1}{3}\big((B-A)+(C-A)\big).$$

The expression above can be interpreted as a step in a Euclidean space from vertex A, in the direction of $\frac{1}{3}\left((B-A)+(C-A)\right)$, which is the arithmetic mean of the directions of vertex A to the three vertices A, B, and C (where direction $A-A$ is trivially omitted). Generalizing the notions of a path (and consequently the direction) between two points and of taking a step in a certain direction to the manifold of positive definite matrices, we obtain the expression (see [9])

$$A \exp\left(\frac{1}{3}\left(\log(A^{-1}B)+\log(A^{-1}C)\right)\right).$$

Hence, in the general case of k matrices, we replace in each iteration the matrices $(A_1^j, \ldots, A_k^j)$ by $(A_1^{j+1}, \ldots, A_k^{j+1})$ in which

$$A_i^{j+1} = A_i^j \exp\left(\frac{1}{k}\sum_{\ell=1,\ell\neq i}^{k} \log\left(\left(A_i^j\right)^{-1} A_\ell^j\right)\right).$$

We iterate until convergence, although convergence is not always guaranteed for this algorithm, i. e., when the matrices are not sufficiently close to each other (see theorem 2.1 in [9] for an exact bound).

For the ALM and NBMP algorithms, the mean of two matrices is by definition known to be the analytical geometric mean, since they are recursively defined, starting with this analytical expression. For the CHEAP mean this consistency is less obvious but it is nonetheless still present. If we examine the CHEAP mean of two matrices by applying one iteration of

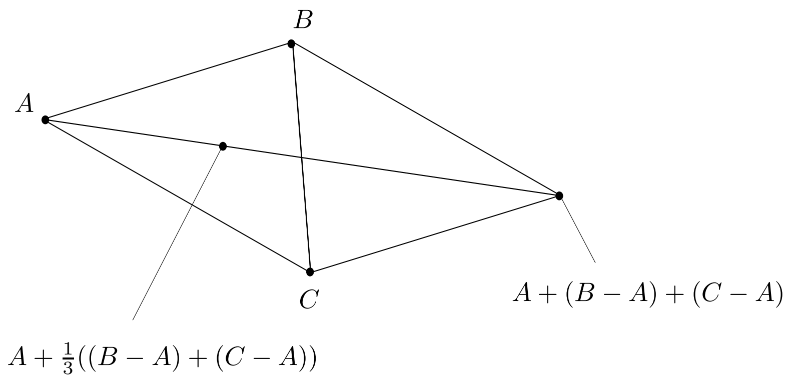


FIGURE 3.2. *Simplified representations of the CHEAP mean for three matrices.*

the algorithm, we get:

$$A_1^{(0)} \to A_1^{(1)} = A_1^{(0)} \exp\left(\frac{1}{2}\log\left((A_1^{(0)})^{-1}A_2^{(0)}\right)\right) = A_1^{(0)}\left((A_1^{(0)})^{-1}A_2^{(0)}\right)^{\frac{1}{2}},$$
$$A_2^{(0)} \to A_2^{(1)} = A_2^{(0)} \exp\left(\frac{1}{2}\log\left((A_2^{(0)})^{-1}A_1^{(0)}\right)\right) = A_2^{(0)}\left((A_2^{(0)})^{-1}A_1^{(0)}\right)^{\frac{1}{2}},$$

which are two equivalent expressions for the geometric mean of $A_1^{(0)}$ and $A_2^{(0)}$.

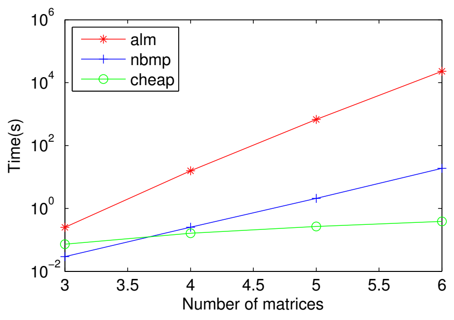
3.5. Comparison. In figure 3.3a, we show the speed of all the above algorithms as the number of 30×30 well-conditioned matrices in the mean increases. The random matrices throughout the paper are constructed in MATLAB as follows:

```
    [Q,~]=qr(rand(n)); D=diag([[rand(1,n-1)+1],10^(-f)]);
    A=Q*D*Q';
```

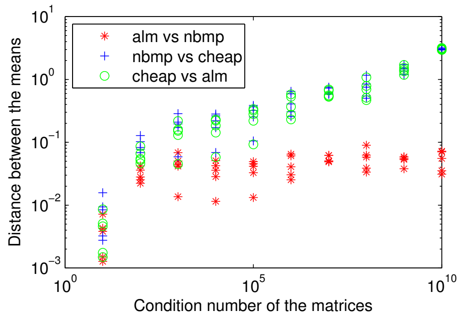
where n is the size of the matrix and f the order of magnitude of the condition number. The stopping criterium for all three algorithms is when the difference between two consecutive iteration points becomes less than a specific tolerance.

While the ALM mean is proven to converge linearly [3] and the NBMP mean superlinearly of order 3 [7], both have rapidly increasing computational time as the number of matrices increases. The number of operations for both algorithms equals $\mathcal{O}(n^3 k! \prod_{i=3}^{k} p_i)$, in which n denotes the size of the matrices, k the number of matrices and p_i the average amount of iterations required to compute the ALM and NBMP mean of i matrices. The advantage of the superlinear convergence of the NBMP algorithm over the linear convergence of the ALM algorithm is found in the p_i factors, since these will be much smaller for the first. The problem for both, however, lies in the significant $k!$ factor, which grows tremendously fast as k increases. Despite the lesser performance, it was still interesting to examine these means since they were the first algorithms devised to compute the matrix geometric mean of a general number of matrices.

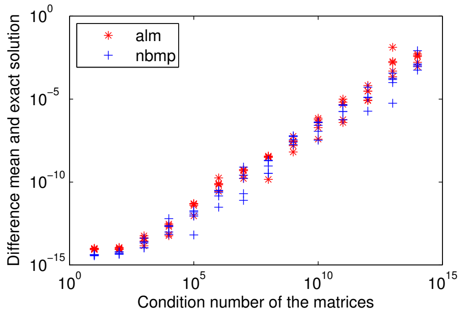
For the CHEAP mean, however, the number of iterations equals $\mathcal{O}(n^3 k^2 p_k)$, in which k^2 is a vast improvement over $k!$. Of course, this increased speed of the CHEAP mean comes at a price. It no longer satisfies all properties in the ALM-list, and can therefore no longer be considered to be an actual geometric mean. We therefore compare the results of the different algorithms by taking the means of three 30×30 matrices of which we vary the condition number. In figure 3.3b, the intrinsic distances (2.2) between the results are shown and it is clear that the ALM and NBMP mean are more similar to each other than to the CHEAP mean,



(a) Required time for different amounts of matrices.



(b) Distance between the different means for the intrinsic measure (2.2).



(c) Accuracy for different conditions of the simultaniously diagonalizable matrices.

FIGURE 3.3. *Comparison of the different planar algorithms.*

especially as the condition of the matrices increases. However, the CHEAP mean can still be found in the vicinity of the other means when the condition of the matrices is acceptable.

A similar figure could be obtained by displaying the classical Euclidean distance between the results, but the distances can be seen to be consistently smaller than for the intrinsic distance (2.2). This difference is explained by the fact that the first is measured on a curved manifold, while the second measures the distance on a straight line in the envelopping vector space $\mathbb{S}^n$, the set of symmetric matrices.

The accuracy of the methods is harder to verify since we need some reference solution to compare the results of the algorithms with. Constructing the matrices in the mean as described above, but with the same matrix Q for all, we obtain a set of simultaniously diagonalizable, and hence commutating matrices. Of these we know the exact geometric mean using the first property in the ALM-list, so we can use this as our reference solution. However, the CHEAP mean is shown [9] to converge in one iteration to the exact solution when the matrices commute. Hence, this test is only of any meaning for the ALM and NBMP mean, of which we show the results in figure 3.3c. The relative intrinsic distance

$$\frac{||\log(A^{-1/2}GA^{-1/2})||_F}{||G||_F}, \tag{3.1}$$

with A the result of one of the algorithms and G the exact solution, is used to display the deviation between both for different conditions of the matrices. Remember that the numerator is the intrinsic distance (2.2) between A and G. We note that when using classical Euclidean distance, the deviations are almost at machine precision for all conditions, and this difference can again be explained by the curvature of the manifold.

4. The Karcher mean. As mentioned in the previous section, the properties in the ALM-list fail to specify a unique definition for the geometric mean. The ALM and NBMP are only two examples of a general class of means satisfying all these properties. Another mean that satisfies all the necessary properties is the Karcher mean. It is defined as the minimizer

$$\mathbf{K}(A_1, ...A_K) = \arg \min_{X \in \mathbb{S}_+^n} \sum_{i=1}^{K} \delta^2(A_i, X),$$

where $\mathbb{S}_+^n$ represents the set of symmetric positive definite matrices, $\delta(A, X)$ is the intrinsic distance on this manifold as given in (2.2) and A_i are the matrices of which we want to find the Karcher mean. In terms of an optimization problem this translates to a cost function f:

$$f(X) = \sum_{i=1}^{K} || \log(A_i^{-1/2} X A_i^{-1/2}) ||_F^2. \tag{4.1}$$

It is well-known that $\mathbb{S}_+^n$ is a convex set and since our cost function is convex as well [5], classical optimization theory assures the existance of a unique (global) minimizer.

This mean satisfies all properties in the ALM-list, of which the monotonicity has only very recently been proven (see [6, 17, 20]). Moreover, the Karcher mean is found to be appealing because of its analogy with the arithmetic mean, which can be seen as a similar minimizer by using the standard Euclidean distance.

Since the Karcher mean will be computed iteratively as the solution of an optimization problem, we need a good starting guess. We will use in all experiments the CHEAP mean, since it possesses good computational speed and reasonable accuracy, as discussed before.

4.1. Differential geometry. Calculating the Karcher mean involves solving an optimization problem on a manifold, which requires a more general approach than in the traditional case of vector spaces. We need to introduce some new concepts to perform this generalization, but we only briefly discuss these matters here; for a more thorough discussion of the subject we refer to any introductory book on differential geometry [11, 18] and to [1] for the optimization perspective. After the introduction, we will use these generalized concepts to implement a number of optimization techniques, more specifically, the steepest descent, conjugate gradient, trust region and BFGS algorithms. The type of generalized optimization used here is often referred to as retraction-based optimization [1, 2], indicating that the concept of retractions (section 4.1.3) lies at the foundation of these techniques.

The general concepts discussed here were found in [1], and many of these structures were already derived for $\mathbb{S}_+^n$ endowed with its natural metric [13, 23, 27]. We added an explicit expression for the Levi–Civita connection (section 4.1.4) on this manifold (and consequently for the Riemannian Hessian) and a derivation of all these structures for the manifold endowed with the inner product inherited from $\mathbb{S}^n$ (see section 4.1.2).

4.1.1. Manifold and tangent space. So far we have been calling the space of positive definite matrices $\mathbb{S}_+^n$ a manifold, without specifying what this means exactly. In order not to get too caught up in details, we give a more intuitive definition: a *manifold* is a set which can locally be mapped one-to-one to $\mathbb{R}^d$ (where d is the dimension of the manifold). In order to get a smooth (C^∞) manifold, we also require these mappings to transition smoothly onto each other in case their domain overlaps. The space $\mathbb{S}_+^n$ is well-known as a smooth manifold [19, 29].

Another important concept is the *tangent space* to a manifold in a certain point, which is basically a first-order (vector space) approximation of the manifold at this point. For $\mathbb{S}_+^n$, the tangent space at each point X, denoted by $T_X\,\mathbb{S}^n_+$, can be identified with the vector space of symmetric matrices $\mathbb{S}^n$

$$T_X\,\mathbb{S}^n_+ \simeq \mathbb{S}^n\,.$$

Applying a tangent vector $\xi_X \in T_X\,\mathbb{S}^n_+$ at a point $X \in \mathbb{S}^n_+$ to a differentiable function $f : \mathbb{S}^n_+ \to \mathbb{R}$ is defined to be

$$\xi_X f = \mathrm{D}\,f(X)[\xi_X]$$

where ξ_X in the right-hand side is simply seen as a symmetric matrix and $\mathrm{D}\,f$ denotes the classical Fréchet derivative of f. For the cost function f in (4.1), this differential is given by [5]

$$\mathrm{D}\,f(X)[\xi_X] = 2\sum_{i=1}^{K} \mathrm{tr}\left(X^{-1}\log(XA_i^{-1})\xi_X\right), \tag{4.2}$$

with A_i the matrices in the mean and $\mathrm{tr}(\cdot)$ the matrix trace.

A *vector field* is a construction that associates with each point on the manifold a tangent vector in its tangent space. Suppose ξ is a vector field on a manifold and f is a real-valued function on this manifold, then ξf is again a real-valued function on the manifold defined by

$$\xi f : \mathbb{S}^n_+ \to \mathbb{R} : X \mapsto \xi_X f.$$

We also apply a more general version of differentiation in this paper, namely of functions between manifolds. This *differential* gives the change of the tangent vectors throughout the function.

4.1.2. Inner product and gradient. Gradient-based optimization requires the notions of a gradient and inner product, which will be introduced here for $\mathbb{S}^n_+$. In fact, we consider two inner products. The first is the inner product most often associated with $\mathbb{S}^n_+$: for $\xi_X, \eta_X \in T_X\,\mathbb{S}^n_+$, we have

$$\langle \xi_X, \eta_X\rangle^{(\mathrm{pd})}_X = \mathrm{tr}(\xi_X X^{-1}\eta_X X^{-1}), \tag{4.3}$$

which leads to the intrinsic distance measure of (2.2) and geodesics of the form of (2.3). Another benefit of this inner product is that the corresponding geodesics are complete, meaning that any geodesic segment can be extended indefinitely. For the second inner product we take the same as the envelopping space $\mathbb{S}^n$: suppose again $\xi_X, \eta_X \in T_X\,\mathbb{S}^n_+$, then

$$\langle \xi_X, \eta_X\rangle^{(\mathrm{sy})}_X = \mathrm{tr}(\xi_X \eta_X). \tag{4.4}$$

As a consequence, the intrinsic distance and expression of the geodesics become the same as in $\mathbb{S}^n$:

$$\delta^{(\mathrm{sy})}(A,B) = ||B-A||_F, \tag{4.5}$$

$$\gamma^{(\mathrm{sy})}(t) = A + t(B-A), \tag{4.6}$$

with $A, B \in \mathbb{S}^n_+$. These geodesics are no longer infinitely extendable since it is possible for some A, B, t that the matrix $\gamma^{(\mathrm{sy})}(t)$ in (4.6) is no longer positive definite and thus not an element of $\mathbb{S}^n_+$. However, for sufficiently small t, $\gamma^{(\mathrm{sy})}(t)$ is in $\mathbb{S}^n_+$ and it could prove to be computationally more efficient than the more involved expression γ^{pd} (2.3).

Further, the gradient of a cost function gives the direction of steepes ascent. It can be defined in each point X as the tangent vector $\operatorname{grad} f(X) \in T_X M$ such that

$$\langle \operatorname{grad} f(X), \xi_X \rangle_X = \mathrm{D}\, f(X)[\xi_X], \quad \forall \xi_X \in T_X M.$$

Using (4.2) we find for our current setting

$$\operatorname{grad}^{(\text{pd})} f(X) = 2 \sum_{i=1}^{K} X^{\frac{1}{2}} \log(X^{\frac{1}{2}} A_i^{-1} X^{\frac{1}{2}}) X^{\frac{1}{2}} = 2 \sum_{i=1}^{K} X \log(A_i^{-1} X), \tag{4.7}$$

when using the inner product in (4.3) and

$$\operatorname{grad}^{(\text{sy})} f(X) = 2 \sum_{i=1}^{K} X^{-\frac{1}{2}} \log(X^{\frac{1}{2}} A_i^{-1} X^{\frac{1}{2}}) X^{-\frac{1}{2}} = 2 \sum_{i=1}^{K} \log(A_i^{-1} X) X^{-1}. \tag{4.8}$$

when using (4.4). Note the slight difference between both in the sign of the power of the outer X-factors.

4.1.3. Retraction and vector transport. Our optimization algorithms require a map, $R_X : T_X \mathbb{S}_+^n \to \mathbb{S}_+^n$, called *retraction*, that locally maps $T_X \mathbb{S}_+^n$ onto the manifold $\mathbb{S}_+^n$ itself, while preserving the first-order information of the tangent space in this point (see figure 4.1a). This means that a step of size zero stays at the same point X and the differential of the retraction in this origin is the identity mapping . An interpretation of these retractions is taking a unit step along a geodesic, or an approximation thereof, on the manifold in the direction specified by the argument. We consider three retractions:

$$R_X^{(\text{sy})}(\xi) = X + \xi, \tag{4.9}$$

$$R_X^{(\text{pd})}(\xi) = X^{\frac{1}{2}} \exp(X^{-\frac{1}{2}} \xi X^{-\frac{1}{2}}) X^{\frac{1}{2}}, \tag{4.10}$$

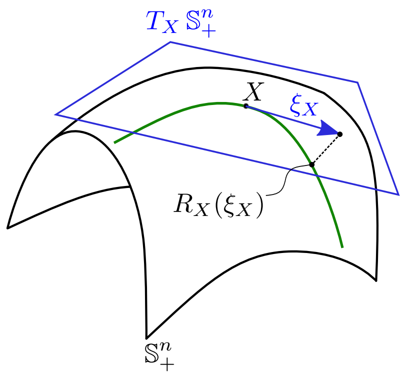
$$R_X^{(\text{pd'})}(\xi) = X + \xi + \frac{1}{2} \xi X^{-1} \xi. \tag{4.11}$$

Note that we omitted the subscript of the tangent vector $\xi \in T_X \mathbb{S}_+^n$ for clarity. The first of these is a unit step along the geodesic (4.6) and can thus be considered to be a natural retraction with respect to inner product (4.4). When the manifold is endowed with inner product (4.3), $R_X^{(\text{sy})}$ is a first-order retraction. As mentioned there, precaution has to be taken to assure that the result of the retraction is still positive definite. We do this by reducing our step size when necessary. The second one is the retraction that naturally arises when the manifold is endowed with the inner product (4.3). Recall that the geodesic between $A, B \in \mathbb{S}_+^n$ is given by

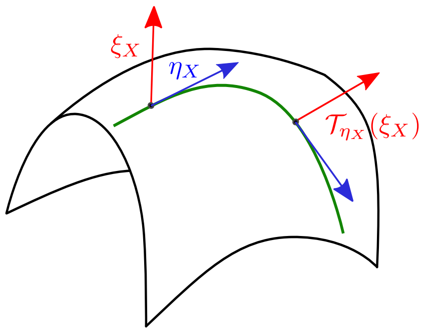
$$\begin{aligned}\gamma(t) &= A^{1/2}(A^{-1/2} B A^{-1/2})^t A^{1/2} \\ &= A^{1/2} \exp\left(t \log(A^{-1/2} B A^{-1/2})\right) A^{1/2}, \quad t \in [0,1].\end{aligned}$$

Now, we obtain (4.10) as $\gamma(t)$ evaluated at $t = 1$ with $\xi_X = A^{1/2} \log(A^{-1/2} B A^{-1/2}) A^{1/2}$ and $A = X$. The last retraction is the second-order approximation to this third retraction, which can easily be seen by using the identity

$$\exp(X) = I + X + \frac{1}{2} X^2 + \mathcal{O}(X^3), \quad X \to 0.$$



(a) Retraction.



(b) Vector transport.

FIGURE 4.1. *Simplified representations of a retraction and a vector transport.*

Next, in order to perform, among others, the conjugate gradient algorithm we need to somehow relate a tangent vector at some point $X \in \mathbb{S}^n_+$ to another $Y \in \mathbb{S}^n_+$. This leads to the concept of a *vector transport* [1] (figure 4.1b). We consider two vector transports: for $X \in \mathbb{S}^n_+, \xi_X, \eta_X \in T_X\,\mathbb{S}^n_+$,

$$\mathcal{T}^{\text{(sy)}}_{\eta_X}(\xi_X) = \xi_X, \tag{4.12}$$

$$\mathcal{T}^{\text{(pd)}}_{\eta_X}(\xi_X) = X^{\frac{1}{2}} \exp\left(\frac{X^{-\frac{1}{2}}\eta_X X^{-\frac{1}{2}}}{2}\right) X^{-\frac{1}{2}}\xi_X X^{-\frac{1}{2}} \exp\left(\frac{X^{-\frac{1}{2}}\eta_X X^{-\frac{1}{2}}}{2}\right) X^{\frac{1}{2}}, \tag{4.13}$$

where $\mathcal{T}^{(.)}_{\eta_X}(\xi_X)$ denotes the vector transport of ξ_X over η_X. The definition of a vector transport (Definition 8.1.1, section 8.1, [1]) states that it needs to be linear in ξ_X and if η_X is the zero element, the vector transport must be the identical mapping. Both these conditions are easily checked for the expressions above. The definition also states that a vector transport has an associated retraction, meaning that the tangent vector $\mathcal{T}^{(.)}_{\eta_X}(\xi_X)$ should be an element of the tangent space at $R_X(\eta_X)$, for some retraction R. Vector transport (4.12) is associated with retraction $R^{\text{(sy)}}_X$ (4.9), since these structures arise naturally arise when is $\mathbb{S}^n_+$ is endowed with the inner product (4.4) of the envelopping vector space $\mathbb{S}^n$. Such a natural vector transport is often refered to as parallel transport. The structure of vector transport (4.13) suggests that it is associated to $R^{\text{(pd)}}_X$ (4.10), which is stated in [13]. Note that it is also possible to find a vector transport associated to $R^{\text{(pd')}}_X$ (4.11), but we decided to restrict our attention to the two more interesting vector transports mentioned above.

4.1.4. Levi–Civita connection and Riemannian Hessian. Some of our optimization methods require second-order information about the system, which is provided by the Hessian operator. The *Riemannian Hessian* of a real-valued function f at a point X on the manifold is a linear, symmetric mapping from the tangent space into itself, given by

$$\operatorname{Hess} f(X) : T_X\,\mathbb{S}^n_+ \to T_X\,\mathbb{S}^n_+ : \xi_X \mapsto \operatorname{Hess} f(X)[\xi_X] = \nabla_{\xi_X} \operatorname{grad} f, \tag{4.14}$$

where ∇ is the so-called *Levi-Civita connection*, which depends on the inner product, hence the Hessian will also depend on the inner product.

When endowed with inner product (4.4), the manifold is a dense Riemannian submanifold of $\mathbb{S}^n$, which is a vectorspace. Hence the Levi–Civita connection is given by

$$\nabla^{\text{(sy)}}_{\zeta_X}\xi = \mathrm{D}(\xi)(X)[\zeta_X],$$

which is simply the derivative as in the vector space.

For the inner product (4.3), however, this connection is more complicated. It can be shown that

$$\nabla^{(\text{pd})}_{\zeta_X}\xi = \mathrm{D}(\xi)(X)[\zeta_X] - \frac{1}{2}\left(\zeta_X X^{-1}\xi_X + \xi_X X^{-1}\zeta_X\right)$$

satisfies all properties of the Levi–Civita connection. A straightforward way to do this, is by checking that it satisfies the Koszul formula [1], which at a point $X \in \mathbb{S}^n_+$ is given by

$$\begin{aligned}2\langle\nabla_{\zeta_X}\eta, \xi_X\rangle_X =& \zeta_X\langle\eta,\xi\rangle + \eta_X\langle\xi,\zeta\rangle - \xi_X\langle\zeta,\eta\rangle \\ & - \langle\zeta_X, [\eta,\xi]_X\rangle_X + \langle\eta_X, [\xi,\zeta]_X\rangle_X + \langle\xi_X, [\zeta,\eta]_X\rangle_X.\end{aligned}$$

The actual computation of the Hessian will be discussed later for each second order method separately.

4.2. First-Order Implementations. We are ready to use all the building blocks of the previous section to assemble a number of optimization methods. These will range from simple first-order techniques such as steepest descent to more advanced second-order methods such as trust region algorithms. In this section, we start by discussing the first-order algorithms while the second-order techniques are examined next.

The basic methods in this section have already been derived in various papers [8, 13, 23, 27], both in their standard form as in some approximated way. As an example of these approximated approaches, we found a Richardson-like iteration in [8], which is derived as the linearization of the standard steepest descent method with inner product (4.3) and retraction (4.10). But this approximation is exactly the steepest descent algorithm using retraction (4.9), hence the technique discussed in the paper can still be interpreted as a steepest descent technique on the manifold.

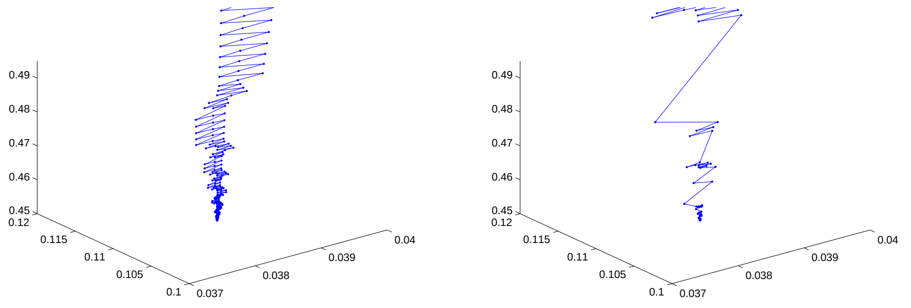
4.2.1. Steepest descent method. In a first attempt, we combine the elements of the previous section into the steepest descent algorithm, which takes in each iteration a step in the direction of $-\operatorname{grad} f(x)$, the direction of steepest descent. The step size is determined using Armijo line search [1], which is a standard backtracking technique, starting from step size 1 and iteratively multiplying it with a factor $\frac{1}{2}$ until an acceptable decrease of the cost function, relative to the step size, is obtained. Algorithm 4.1 contains the steepest descent method when $\beta = 0$. The convergence conditions checked in this algorithm are whether the Armijo step size or the absolute or relative difference between two consecutive iterations are smaller than their respective tolerances. We consider all three retractions (4.9), (4.10) and (4.11).

4.2.2. Conjugate gradient method. In figure 4.2a, we show the typical zigzag-pattern that arises for the steepest descent method. Algorithm 4.1 shows the conjugate gradient algorithm which helps us deal with this problem, as can be seen in figure 4.2b. The amount of influence of the previous search direction in the conjugate gradient algorithm is determined by the β factor, for which we consider three different formulas, given in [22] and denoted by $\beta^{(\text{fr})}$ (Fletcher–Reeves), $\beta^{(\text{pr})}$ (Polak–Ribière) and $\beta^{(\text{hs})}$ (Hestenes–Stiefel). As mentioned, this previous search direction is transported between different tangent spaces by vector transports. To get the best affinity between the vector transports and the retractions, we work with the natural retractions $R^{(\text{sy})}$ (with corresponding vector transport $\mathcal{T}^{(\text{sy})}$) and $R^{(\text{pd})}$ (with $\mathcal{T}^{(\text{pd})}$) here.

4.2.3. Comparison. When comparing the overall performance of the steepest descent and conjugate gradient algorithms, we notice that the influence of choosing inner product

Algorithm 4.1 The Karcher mean using the conjugate gradient method

```
Input: matrices $A_1, ..., A_K$, $K > 2$, initial guess $X_0$, retraction and vector transport type $R^i$
  and $\mathcal{T}^i$ ($R^{(sy)}$ and $\mathcal{T}^{(sy)}$ or $R^{(pd)}$ and $\mathcal{T}^{(pd)}$), $\beta$ type $\beta^j$ ($\beta^{(fr)}$, $\beta^{(pr)}$ or $\beta^{(hs)}$ or 0)
Output: Karcher mean $\mathbf{K}(A_1, ..., A_K)$
  $k \leftarrow 0$
  $grad_k \leftarrow \operatorname{grad} f(X_k)$                          { $f$ is our cost function }
  $\xi_k \leftarrow -grad$                                   {$\xi$ is the search direction}
  while not converged do
    $X_{k+1} \leftarrow R^i_{X_k}(t^A \xi_k)$                   { with $t^A$ the Armijo step size [1]}
    $grad_{k+1} \leftarrow \operatorname{grad} f(X_{k+1})$
    $\xi_{old} \leftarrow \mathcal{T}^i_{t^A \xi_k}(\xi_k)$         {vector transport of the old search direction}
    Determine $\beta$ according to the given type
    $\xi_{k+1} \leftarrow -grad_{k+1} + \beta \xi_{old}$
    if $\xi_{k+1}$ not a descent direction then
      $\xi_{k+1} \leftarrow -grad_{k+1}$
    end if
    $k \leftarrow k + 1$
  end while
```

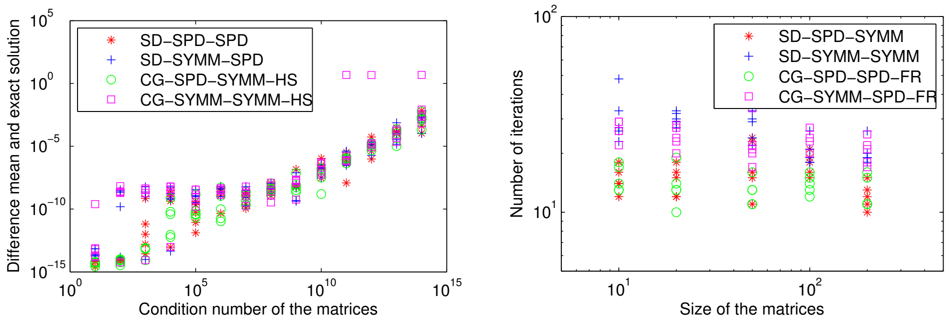


(a) Using steepest descent. (b) Using conjugate gradient.

FIGURE 4.2. *Evolution of the eigenvalues of the consecutive iteration points in calculating the Karcher mean of five* 3×3 *matrices for which the zigzag-pattern appears for steepest descent.*

(4.3) or (4.4) is far greater than the impact of the chosen retraction or, in case of the conjugate gradient algorithm, the β type. In fact, when only varying the retraction and β type, the results are all very similar. The speed-up of the conjugate gradient technique over steepest descent is also hardly noticeable in general, which is explained by the presence of a sufficiently good initial guess, the CHEAP mean. When this initial point is sufficiently close to the solution of the problem, the cost function will behave nicely in this neighbourhood and the zigzag-pattern mentioned before is less likely to occur.

In figure 4.3a, the relative intrinsic distance (3.1) between the results of the algorithms and the exact solution is given as function of the condition of the matrices. The exact solution is again determined by choosing the three 30×30 matrices in the mean to be simultaniously diagonalizable, causing the exact solution to be known analytically (the matrices are again constructed as discussed in section 3.5). For lower condition numbers, the algorithms using the natural inner product (4.3) display better results in general than those using (4.4). As the



(a) Accuracy for different conditions numbers.

(b) Required iterations for different sizes.

FIGURE 4.3. *Comparison of the accuracy and computational speed of algorithm 4.1 using the steepest descent and conjugate gradient algorithm. In the legends, we first indicate whether the Steepest Descent (SD) or Conjugate Gradient (CG) technique is used, next, which of the inner products (4.3) (SPD) or (4.4) (SYMM) is used and finally whether retraction $R_X^{(sy)}$ (SYMM) or $R_X^{(pd)}$ (SPD) was taken. In case of CG, we also indicate which β type is used.*

condition number grows, the results start deteriorating equally, which as in section 3.5 can be explained by the increased curvature of the manifold towards its boundary.

The speed of the algorithms is tested both for an increasing number of matrices (with the size fixed to 10×10 matrices) and for varying sizes of the matrices in the mean (with the amount fixed to five). Again we notice the advantage of the structures associated with inner product (4.3), as these result in algorithms requiring less iterations. The computational cost of these structures, however, is usually higher than those related to inner product (4.4), which causes the overall computational time to be very similar. In figure 4.3b, the number of iterations is displayed for the algorithms as a function of the size of the matrices, which clearly shows the distinction between the two inner products.

We can conclude that structures associated with inner product (4.3) are best suited for our problem since the resulting algorithms require less iterations. It therefore seems interesting to consider the performance of a steepest descent algorithm when the inner product

$$\langle \xi_X, \eta_X \rangle_X^{(\alpha)} = \operatorname{tr}\left(\xi_X X^{-\alpha} \eta_X X^{-\alpha}\right) \tag{4.15}$$

and its related structures are used, which reduces to the previous cases when $\alpha = 0$ or $\alpha = 1$. In figure 4.4, we display the number of iterations such an algorithm requires for different values of α. The figure displayed has been constructed using the retraction $R_X^{(\mathrm{pd})}$, but a nearly identical figure was obtained using $R_X^{(\mathrm{sy})}$, again indicating that the influence of the inner product is far greater than that of the retraction. It is obvious from the figure that inner product (4.3) is still most natural to the manifold, since this is the one corresponding to $\alpha = 1$.

4.3. Second-Order Implementations. The goal of second-order optimization techniques is to use (an approximation of) the Hessian of the cost function to obtain a quadratic (or at least superlinear) convergent algorithm. In the following, we discuss a number of attempts to accomplish this and compare their performance. This Hessian has, however, typically a higher computational complexity, which means it is yet to be determined whether these algorithms are more efficient than the first-order techniques. In the discussions of the trust region method, we focus on the computation of the Hessian while the actual implementation of the method is performed using algorithms 10 and 11 from [1].

In existing literature, the Riemannian Hessian is computed as defined in section 4.3.2 [12, 27]. We derive the Hessian according to the classical definition (4.14) (using the Levi–Civita connection) in section 4.3.1 and an approximation in section 4.3.3. The Riemannian

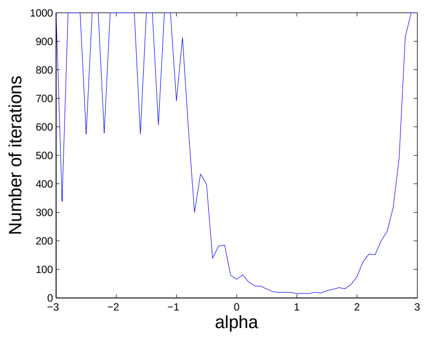


FIGURE 4.4. *Number of iterations required for a steepest descent algorithm when using structures related to inner product* (4.15) *for different values of* α.

BFGS method in section 4.3.4 is the application to $\mathbb{S}_+^n$ of the existing generalized algorithm [25, 28].

4.3.1. Trust region method: Exact Hessian. In section 4.1.4, we derived all the components needed to determine the Hessian of our cost function f and noticed that the result again depends on the inner product. Hence using the definition of the Hessian for the inner product (4.4), with $\nabla^{(\mathrm{sy})}$, gives

$$\mathrm{Hess}^{(\mathrm{sy})} f(X)[\xi_X] = \mathrm{D}(\mathrm{grad}^{(\mathrm{sy})} f)(X)[\xi_X].$$

Using (4.8) leads to

$$\begin{aligned}\mathrm{Hess}^{(\mathrm{sy})} f(X)[\xi_X] =& 2\sum_{i=1}^{K} \mathrm{D}(\log)(A_i^{-1}X)[A_i^{-1}\xi_X]X^{-1} \\ &- 2\sum_{i=1}^{K} \log(A_i^{-1}X)X^{-1}\xi_X X^{-1},\end{aligned}$$

where we used the product and chain rules of differentiation as well as the differential of the matrix inverse function. We also recognize the differential of the matrix logarithm function at point $A_i^{-1}X$ in the direction of tangent vector $A_i^{-1}\xi_X$, which can be calculated using algorithm 11.12 in [16].

When using the inner product (4.3) with $\nabla^{(\mathrm{pd})}$, the Hessian becomes

$$\begin{aligned}\mathrm{Hess}^{(\mathrm{pd})} f(X)[\xi_X] =& \mathrm{D}(\mathrm{grad}^{(\mathrm{pd})} f)(X)[\xi_X] \\ &- \frac{1}{2}\left(\xi_X X^{-1}\,\mathrm{grad}^{(\mathrm{pd})} f(X) + \mathrm{grad}^{(\mathrm{pd})} f(X) X^{-1}\xi_X\right).\end{aligned} \tag{4.16}$$

In this case using (4.7) leads to

$$\begin{aligned}\operatorname{Hess}^{(\mathrm{pd})} f(X)[\xi_X] =& 2\sum_{i=1}^{K} \xi_X \log(A_i^{-1}X) + 2\sum_{i=1}^{K} X\,\mathrm{D}(\log)(A_i^{-1}X)[A_i^{-1}\xi_X] \\ &- \left(\sum_{i=1}^{K} \xi_X \log(A_i^{-1}X) + \sum_{i=1}^{K} \log(XA_i^{-1})\xi_X\right) \\ =& \sum_{i=1}^{K} \xi_X \log(A_i^{-1}X) - \sum_{i=1}^{K} \log(XA_i^{-1})\xi_X \\ &+ 2\sum_{i=1}^{K} X\,\mathrm{D}(\log)(A_i^{-1}X)[A_i^{-1}\xi_X],\end{aligned} \tag{4.17}$$

where again we need the differential of the matrix logarithm. Note that the first two terms in (4.17) are each others transpose (except the minus sign), which can be exploited in its computation. At first sight this can seem somewhat peculiar, since this subtraction produces a skew-symmetric matrix while the result of the Hessian is supposed to be symmetric. However, looking at (4.16), we can see that the differential of the gradient is symmetric and the second part, as the sum of a matrix and its transpose, is symmetric as well, proving that $\operatorname{Hess}^{(\mathrm{pd})} f(X)[\xi_X]$ is an element of $\mathbb{S}^n$.

4.3.2. Trust region method: Hessian by decomposition. Another way to compute the Hessian of our cost function is described in [12], and more explicitly for the current cost function in [27]. The proposed procedure consists of determining the components of a decomposition and then combining these to form the actual Hessian. A downside, however, is that when determining the Karcher mean of K $n \times n$ matrices, this sum consists of $Kn(n+1)/2$ terms, which grows rapidly as n increases (in [27], only 3×3 matrices were considered). Two other important remarks are that this technique is derived for the manifold endowed with the inner product (4.3) and that the computation of the terms as in [27] is only valid when we take the Hessian at the identity matrix $X = I$. This causes the need to translate the problem in each iteration step to ensure this position for the current iteration point, which is done by applying the mapping $Y \mapsto X_k^{-1/2} Y X_k^{-1/2}$, in which X_k is the newly found iteration point, to all matrices in the mean and to X_k itself. After convergence, we apply the inverse mapping to translate the identity matrix to the actual Karcher mean of the original matrices. Theoretically, this need for a translation is not a downside, in fact, it can even simplify notations and required structures. Computationally, however, it could cause problems when working, e. g., with ill-conditioned matrices.

The expression for the Hessian is given by

$$\begin{aligned}\operatorname{Hess}^{(\mathrm{pd})} f(X)[\xi_X] =& 2\sum_{l=1}^{m} \langle \xi_X, E_l\rangle_X w_l(1) E_l \\ =& 2\sum_{l=1}^{m} \operatorname{tr}(\xi_X X^{-1} E_l X^{-1}) w_l(1) E_l,\end{aligned}$$

where $m = Kn(n+1)/2$ and E_l, w_l are defined as in [27]. Remember that the decomposition is only valid at $X = I$, hence the expression can be further simplified to

$$\operatorname{Hess}^{(\mathrm{pd})} f(X)[\xi_X] = 2\sum_{l=1}^{m} \operatorname{tr}(\xi_X E_l) w_l(1) E_l. \tag{4.18}$$

When computing the Hessian, the rank one structure of E_l can be exploited to limit the required amount of operations. In fact, if we compare the cost to compute the Hessians (4.17) and (4.18), which give the same Hessian, we find the amount of operations for the first to be $\mathcal{O}(Kn^3)$ and that of the second $\mathcal{O}(Kn^4)$. This first amount, however, is accompanied by a very large constant factor of well over 1000, which means n needs to be larger than 1000 before we can actually see a speed-up of (4.17) over (4.18).

4.3.3. Trust region method: Hessian by approximation. In the calculations of the exact Hessian, determining the differential of the matrix logarithm function appears to be the main bottleneck in terms of computational efficiency. A perhaps less elegant, but sometimes advantageous solution is to replace the matrix logarithm by its Taylor series, at $X = I$ (with I the identity matrix) given by

$$\sum_{m=1}^{\infty} \frac{(-1)^{m+1}}{m}(X-I)^m,$$

which converges to $\log(X)$ for all $\rho(X-I) < 1, X \neq 0$, where ρ denotes the spectral radius. Hence when the matrices in the mean lie close to each other (and to the current estimate), this technique is expected to work well. Truncating this series after the second term and entering the result into the expressions for the gradient (4.7) and (4.8), corresponding to inner products (4.3) and (4.4), we obtain

$$\begin{aligned}
\operatorname{grad}_2^{(\mathrm{pd})} f(X) &= 2\sum_{i=1}^{K}(2XA_i^{-1}X - \frac{3}{2}X - \frac{1}{2}XA_i^{-1}XA_i^{-1}X), \text{ and} \\
\operatorname{grad}_2^{(\mathrm{sy})} f(X) &= 2\sum_{i=1}^{K}(2A_i^{-1} - \frac{3}{2}X^{-1} - \frac{1}{2}A_i^{-1}XA_i^{-1})
\end{aligned}$$

respectively as an approximation to the gradient. Note that we only use this approximation to derive the approximated Hessian, while the actual gradient will be used in the final algorithm. Applying the definition of the Hessian to these expressions results in

$$\begin{aligned}
\operatorname{Hess}_2^{(\mathrm{pd})} f(X)[\xi_X] &= \mathrm{D}(\operatorname{grad}_{spd,2} f)(X)[\xi_X] \\
&\quad - \frac{\xi_X X^{-1}\operatorname{grad}_2^{(\mathrm{pd})} f(X) + \operatorname{grad}_2^{(\mathrm{pd})} f(X)X^{-1}\xi_X}{2} \\
&= \sum_{i=1}^{K}\Bigg(2XA_i^{-1}\xi_X + 2\xi_X A_i^{-1}X - \frac{1}{2}\xi_X A_i^{-1}XA_i^{-1}X \\
&\quad -XA_i^{-1}\xi_X A_i^{-1}X - \frac{1}{2}XA_i^{-1}XA_i^{-1}\xi_X\Bigg), \\
\operatorname{Hess}_2^{(\mathrm{sy})} f(X)[\xi_X] &= \mathrm{D}(\operatorname{grad}_2^{(\mathrm{sy})} f)(X)[\xi_X] \\
&= \sum_{i=1}^{K}\left(3X^{-1}\xi_X X^{-1} - A_i^{-1}\xi_X A_i^{-1}\right).
\end{aligned}$$

Since this is only an approximation to the Hessian, quadratic convergence of the algorithm is no longer guaranteed as will be shown in section 4.3.5.

4.3.4. Riemannian BFGS method. Finally, we test a Riemannian generalization of the classical BFGS method [25, 28], which updates an estimation of the Hessian throughout the algorithm, instead of solving systems with the Hessians exactly. The main point of interest is how this update is performed. Suppose we know the estimate $\mathcal{B}_k$ in iteration step k, which is assumed to be a linear operator from $T_{X_k}\,\mathbb{S}_+^n$ onto itself and can thus be represented by an $(n(n+1)/2) \times (n(n+1)/2)$ matrix B_k. Then the linear operator $\mathcal{B}_{k+1} : T_{X_{k+1}}\,\mathbb{S}_+^n \to T_{X_{k+1}}\,\mathbb{S}_+^n$ is defined by

$$\mathcal{B}_{k+1}p = \widetilde{\mathcal{B}}_k p - \frac{\langle s_k, \widetilde{\mathcal{B}}_k p\rangle_{X_{k+1}}}{\langle s_k, \widetilde{\mathcal{B}}_k s_k\rangle_{X_{k+1}}}\widetilde{\mathcal{B}}_k s_k + \frac{\langle y_k, p\rangle_{X_{k+1}}}{\langle y_k, s_k\rangle_{X_{k+1}}} y_k, \qquad \forall p \in T_{X_{k+1}}\,\mathbb{S}_+^n \tag{4.19}$$

$$\widetilde{\mathcal{B}}_k = \mathcal{T}_{t^A\eta_k} \circ \mathcal{B}_k \circ \left(\mathcal{T}_{t^A\eta_k}\right)^{-1}, \tag{4.20}$$

in which η_k is the current search direction, t^A is the Armijo step size, s_k is the vector transport of search direction to the new iteration point, y_k is a measure for the change of the gradient over the iteration step (formal expressions can be found in algorithm 4.2), and $\mathcal{T}$ is the vector transport. Since an inner product and a vector transport are present in these expressions, there are again different situations to investigate. We will test the algorithm for each of the two inner products, combined with their natural retraction and vector transport. Note that to evaluate $\mathcal{B}_{k+1}p$ using the matrix representation B_{k+1}, we need an $n(n+1)/2$ vector representation of p. This is done by the half-vectorization operator vech, which stacks the elements of the upper triangular part of p columnwise. Using this representation, the matrix-vector product $B_{k+1}\,\mathrm{vech}(p)$ returns an $(n(n+1)/2)$ vector which is the half-vectorization of the matrix $\mathcal{B}_{k+1}p$.

In the simpler case of inner product (4.4), expressions (4.19) and (4.20) become

$$\begin{aligned} B_{k+1}\,\mathrm{vech}(p) =& \widetilde{B}_k\,\mathrm{vech}(p) - \widetilde{B}_k\,\mathrm{vech}(s_k)\frac{\mathrm{tr}(s_k\widetilde{\mathcal{B}}_k p)}{\mathrm{tr}(s_k\widetilde{\mathcal{B}}_k s_k)} \\ &+ \mathrm{vech}(y_k)\frac{\mathrm{tr}(y_k p)}{\mathrm{tr}(y_k s_k)}, \qquad \forall p \in T_{X_{k+1}}\,\mathbb{S}_+^n \\ \widetilde{\mathcal{B}}_k =& \mathcal{B}_k. \end{aligned}$$

To remove p from this expression, the matrix traces need to be split using the property $\mathrm{tr}(AB) = \mathrm{vec}(A)^T\,\mathrm{vec}(B)$ with A and B symmetric matrices. For the expression above, however, the second matrix should be half-vectorized, which needs to be compensated for in the first vectorization. To this end, we also change the vectorization of the first matrix to half-vectorization, but with the adaptation that each off-diagonal element is doubled and this operation is denoted by vech_2. This yields $\mathrm{tr}(AB) = \mathrm{vech}_2(A)^T\,\mathrm{vech}(B)$ and our update formula becomes

$$\begin{aligned} B_{k+1} =& B_k - \frac{1}{\mathrm{tr}(s_k\mathcal{B}_k s_k)} B_k\,\mathrm{vech}(s_k)\,\mathrm{vech}_2(s_k)^T B_k \\ &+ \frac{1}{\mathrm{tr}(y_k s_k)}\,\mathrm{vech}(y_k)\,\mathrm{vech}_2(y_k)^T. \end{aligned} \tag{4.21}$$

For inner product (4.3), the calculation of $\widetilde{\mathcal{B}}_k$ is no longer so straightforward. Entering vector transport $\mathcal{T}^{(\mathrm{pd})}$ (4.13) into equation (4.20), we obtain

$$\widetilde{\mathcal{B}}_k p = Q\mathcal{B}_k(Q^{-1}pQ^{-T})Q^T, \qquad Q = X_k^{\frac{1}{2}}\exp\left(\frac{X_k^{-\frac{1}{2}}t^A\eta_k X_k^{-\frac{1}{2}}}{2}\right)X_k^{-\frac{1}{2}}.$$

To extract p from this expression, we want to use the property $\mathrm{vec}(ABC) = (C^T \otimes A)\,\mathrm{vec}(B)$ for general matrices A, B, and C, with $\otimes$ the Kronecker product, but this property cannot be used when half-vectorization is applied. Therefore, let us pretend for a moment that $\mathcal{B}_k$ and $\widetilde{\mathcal{B}}_k$ are represented by $n^2 \times n^2$ matrices and apply this rule to the above expression:

$$\begin{aligned}\widetilde{B}_k \operatorname{vec}(p) &= \operatorname{vec}(Q\mathcal{B}_k(Q^{-1}pQ^{-T})Q^T) \\ &= (Q \otimes Q)B_k \operatorname{vec}(Q^{-1}pQ^{-T}) \\ &= (Q \otimes Q)B_k(Q^{-1} \otimes Q^{-1}) \operatorname{vec}(p).\end{aligned}$$

Changing back to half-vectorization can be accomplished by using the so-called duplication and elimination matrices D_n and E_n, which are simple matrices for transforming respectively a half-vectorization into a normal vectorization and vice versa. Equation (4.19) can be tackled in the same fashion as before, where we only need to pay attention to the extra factors in the current inner product. The total update procedure now becomes

$$\begin{aligned}\widetilde{B}_k &= E_n(Q \otimes Q)D_nB_kE_n(Q^{-1} \otimes Q^{-1})D_n, \\ B_{k+1} &= \widetilde{B}_k - \frac{1}{\operatorname{tr}(X_{k+1}^{-1}s_kX_{k+1}^{-1}\widetilde{\mathcal{B}}_ks_k)}\widetilde{B}_k \operatorname{vech}(s_k) \operatorname{vech}_2(X_{k+1}^{-1}s_kX_{k+1}^{-1})^T\widetilde{B}_k \\ &\quad + \frac{1}{\operatorname{tr}(X_{k+1}^{-1}y_kX_{k+1}^{-1}s_k)} \operatorname{vech}(y_k) \operatorname{vech}_2(X_{k+1}^{-1}y_kX_{k+1}^{-1})^T.\end{aligned} \tag{4.22}$$

Now that the techniques to update the estimate of the Hessian are specified, we show the entire Riemannian BFGS method in algorithm 4.2. The convergence criteria consist as before of whether the Armijo step size or the absolute or relative difference between two consecutive iterations are smaller than their respective tolerances.

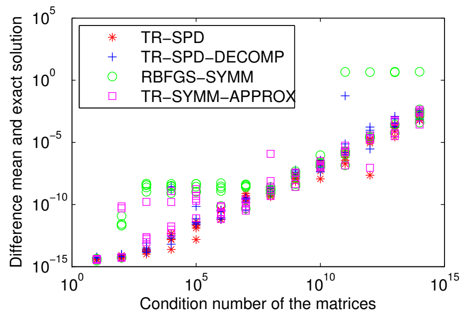
Algorithm 4.2 The Karcher mean using the Riemannian BFGS method

Input: matrices $A_1, ..., A_K$, $K > 2$, initial guess X_0, initial Hessian approximation B_0, retraction and vector transport type R^i and $\mathcal{T}^i$ ($R^{(\mathrm{sy})}$ and $\mathcal{T}^{(\mathrm{sy})}$ or $R^{(\mathrm{pd})}$ and $\mathcal{T}^{(\mathrm{pd})}$)

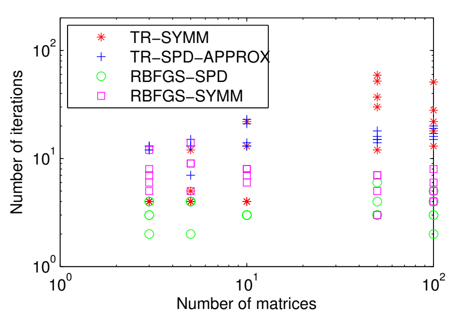
Output: Karcher mean $\mathbf{K}(A_1, ..., A_K)$

```
$k \leftarrow 0$
$grad_k \leftarrow \operatorname{grad} f(X_k)$                                  {$f$ is our cost function}
while not converged do
    Obtain $\eta_k$: Solve the system: $B_k \operatorname{vech}(\eta_k) = -\operatorname{vech}(grad)$      {The search direction}
    if $\eta_k$ not a descent direction then
        $\eta_k \leftarrow -grad_k$
    end if
    $X_{k+1} \leftarrow R^i_{X_k}(t^A\eta_k)$                          {with $t^A$ the Armijo step size [1]}
    $grad_{k+1} \leftarrow \operatorname{grad} f(X_{k+1})$
    $s_k \leftarrow \mathcal{T}^i_{t^A\eta_k}(t^A\eta_k)$
    $y_k \leftarrow grad_{k+1} - \mathcal{T}^i_{t^A\eta_k}(grad_k)$
    Update $B_k$ to $B_{k+1}$ using (4.21) or (4.22), depending on the inner product
    $k \leftarrow k + 1$
end while
```

4.3.5. Comparison. We start again by applying the second-order techniques to three simultaniously diagonalizable, 30×30 matrices and compare the results with the exact solution, which we display in figure 4.5a. The methods perform with accuracy similar to that of the steepest descent and conjugate gradient methods, where we notice again slightly better results for the techniques corresponding to inner product (4.3). This lesser accuracy for



(a) Accuracy for different conditions of the matrices.



(b) Required iterations for a different number of matrices.

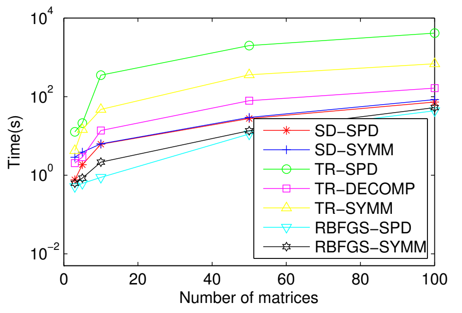
FIGURE 4.5. *Comparison of the accuracy and computational speed of the Karcher mean using the various trust region methods and the Riemannian BFGS algorithm. In the legends, we first indicate whether the Trust Region (TR) or Riemannian BFGS (RBFGS) technique is used, next, which of the inner products (4.3) (SPD) or (4.4) (SYMM) is used and finally an extra term if a non-standard method was used (DECOMP for the technique in section 4.3.2 and APPROX for those in section 4.3.3).*

inner product (4.4) is most noticeable for the Riemannian BFGS method and the trust region algorithms using the approximated Hessian, as shown in the figure.

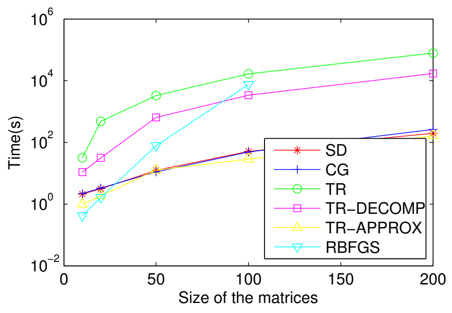
To test the speed of the algorithms, the size of the matrices in the mean is again varied (when taking 5 matrices) as well as the number of matrices (where we fix the size to 10×10 matrices). We notice once more the lesser amount of iterations required by techniques based on inner product (4.3), although the corresponding structures will in general be more expensive to compute. In figure 4.5b, the number of iterations are shown for some of the methods as the number of matrices in the mean varies, where the difference in iterations for the RBFGS methods displays our statement. We also noticed that as the number of matrices increases, the convergence of the trust region method with inner product (4.4) fails to be quadratic. This instability is also noticed for the trust region method using inner product (4.3) and the same phenomena occurs as the size of the matrices increases. The higher computational cost when using inner product (4.3) can clearly be seen when comparing the computational time of the two resulting trust region methods (see figure 4.6a: TR-SPD and TR-SYMM).

Another remarkable result in figure 4.6a is the performance of the Riemannian BFGS method, which displays a lower computational time than the steepest descent algorithm. We do note that this test is performed for a varying number of 10×10 matrices, and as the size of the matrices starts to increase, the Riemannian BFGS method, as well as all second-order techniques, are quickly outperformed by the steepest descent and conjugate gradient algorithm (see figure 4.6b). Figure 4.6a also shows that the Riemannian BFGS method based on inner product (4.3) is faster than the one using inner product (4.4), even though the corresponding structures are more expensive. As figure 4.5b showed, the number of iterations is significantly lower for the first, which makes up for this extra computational cost. However, we note again that as the size of the matrices increases, the method using inner product (4.4) becomes faster due to its less expensive update formula.

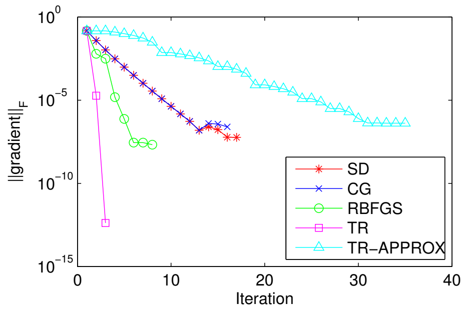
Further, we note that the technique in section 4.3.2 is an improvement over the classical method using the standard definition of the Hessian (4.16) due to the amount of operations for both Hessians we discussed in section 4.3.3. As discussed there, the Hessian (4.17) would be cheaper to compute if the size of the matrices is at least over 1000, but we only tested sizes up to $n = 200$. Testing for larger matrices would not be useful since both trust region methods are outperformed by the first-order techniques here. As mentioned before, the method using



(a) Required time for different methods using the two inner products for varying number of matrices.



(b) Required time for different methods using inner product (4.3) for varying sizes.



(c) Norm of the gradient throughout the iterations.

FIGURE 4.6. *Comparison of all the discussed algorithms. In the legends, the abbreviations SD (Steepest Descent), CG (Conjugate Gradient), TR (Trust Region) and RBFGS (Riemannian BFGS) are used to denote the applied technique and when necessary, the used inner product is indicated with SPD* (4.3) *or SYMM* (4.4)*. For TR the suffix DECOMP or APPROX is added to indicate the techniques in section 4.3.2 and 4.3.3 respectively. In the first figure, the influence of the inner products is compared. In the second, we compare the algorithms which use inner product* (4.3)*. In the last figure, the evolution of the gradient is depicted for all techniques using inner product* (4.3)*.*

(4.17) also has some problems concerning stability as well, since convergence seems to fail as the number of matrices or their sizes increase.

Figure 4.6c shows the evolution of the gradient for all algorithms based on inner product (4.3). The quadratic convergence of the trust region algorithm is clearly visible, as well as a superlinear convergence for the Riemannian BFGS method. The steepest descent and conjugate gradient algorithm display very similar (linear) convergence since the problem is well-behaved, eliminating the need for the conjugate gradient technique to be activated. Finally, the trust region algorithm using the approximated Hessian (section 4.3.3) has lost all quadratic convergence and displays an even slower convergence than the steepest descent method. We note that the techniques that use the Armijo line search technique to determine the next iterations point stop when the norm of the gradient is about the square root of the machine precision. This is caused by the use of the squared norm of the gradient in the Armijo condition.

5. Conclusions. This paper has demonstrated various techniques to compute a matrix geometric mean. The lack of uniqueness of the definition was mostly overcome by the appealing analogy of the Karcher mean with the arithmetic mean. The convergence of the first-order optimization techniques for computing this Karcher mean can easily be verified using corollary 4.3.2 and theorems 4.4.1 and 4.4.2 in [1] and exploiting the convexity of the problem. The convergence of the second-order optimization methods, although predicted by the experiments, is theoretically not so easily guaranteed and will be treated as future work (indications are present in [1, 15]).

We noticed that while the second-order techniques required less iterations, the computational cost associated with each of these iterations was much higher than that of the first-order algorithms, nullifying the advantage of quadratic convergence. Hence we conclude that for the current algorithms on the manifold $\mathbb{S}_+^n$, it is more advantageous to work with first-order optimization techniques when the size of the matrices increases (already at $n = 10$). It is possible to produce more efficient second-order optimization algorithms if we were to reduce our search space, so the manifold of interest, to a certain subset of which the structure can be further exploited. For example, the geometry of the manifold of larger matrices of fixed, low rank has already been extensively researched [10, 30], and can be used to apply the optimization techniques in this paper. This will also be a topic of future research.

The MATLAB code used to produce the experiments in this paper will be made available on the research page of Prof. Dr. Raf Vandebril[1].

Acknowledgements. This research was partially supported by the Research Council K.U.Leuven, projects OT/11/055 (Spectral Properties of Perturbed Normal Matrices and their Applications), CoE EF/05/006 Optimization in Engineering (OPTEC), by the Fund for Scientific Research–Flanders (Belgium) project G034212N (Reestablishing smoothness for matrix manifold optimization via resolution of singularities) and by the Interuniversity Attraction Poles Programme, initiated by the Belgian State, Science Policy Office, Belgian Network DYSCO (Dynamical Systems, Control, and Optimization). We would also like to thank Bruno Iannazzo for the fruitful discussions and updates about new developments.

REFERENCES

[1] P.-A. ABSIL, R. MAHONY, AND R. SEPULCHRE, *Optimization Algorithms on Matrix Manifolds*, Princeton University Press, 2008.

[2] R. L. ADLER, J.-P. DEDIEU, J. Y. MARGULIES, M. MARTENS, AND M. SHUB, *Newton's method on Riemannian manifolds and a geometric model for the human spine*, IMA Journal of Numerical Analysis, 22 (2002), pp. 359–390.

[3] T. ANDO, C. LI, AND R. MATHIAS, *Geometric means*, Linear Algebra and its Applications, 385 (2004), pp. 305–334.

[4] F. BARBARESCO, *New foundation of radar Doppler signal processing based on advanced differential geometry of symmetric spaces: Doppler matrix CFAR & radar application*, in International Radar Conference 2009, Bordeaux, France, SIAM, October 2009.

[5] R. BHATIA, *Positive Definite Matrices*, Princeton Series in Applied Mathematics, Princeton University Press, 2007.

[6] R. BHATIA AND R. L. KARANDIKAR, *The matrix geometric mean*. Preprint at www.isid.ac.in/~statmath/eprints, 2011.

[7] D. BINI, B. MEINI, AND F. POLONI, *An effective matrix geometric mean satisfying the Ando–Li–Mathias properties*, Mathematics of Computation, 79 (2010), pp. 437–452.

[8] D. A. BINI AND B. IANNAZZO, *Computing the Karcher mean of symmetric positive definite matrices*, tech. report. To appear.

[9] D. A. BINI AND B. IANNAZZO, *A note on computing matrix geometric means*, Advances in Computational Mathematics, 32 (2010), pp. 1–18.

[10] S. BONNABEL AND R. SEPULCHRE, *Geometric distance and mean for positive semi-definite matrices of fixed rank*, SIAM Journal on Matrix Analysis and Applications, (2009).

[11] W. M. BOOTHBY, *An introduction to differentiable manifolds and Riemannian geometry*, Academic Press, Inc., 1975.

[12] R. FERREIRA, J. XAVIER, J. COSTEIRA, AND V. BAROSSO, *Newton algorithms for Riemannian distance related problems on connected locally symmetric manifolds*, tech. report, Institute for Systems and Robotics (ISR), Signal and Image Processing Group (SPIG), Instituto Superior Técnico (IST), 2008.

[1] http://people.cs.kuleuven.be/~raf.vandebril/

[13] R. FERREIRA, J. XAVIER, J. COSTEIRA, AND V. BARROSO, *Newton method for Riemannian centroid computation in naturally reductive homogeneous spaces*, in IEEE International Conference on Acoustics, Speech and Signal Processing, IEEE, 2006.

[14] P. FLETCHER AND S. JOSHI, *Riemannian geometry for the statistical analysis of diffusion tensor data*, Signal Processing, 87 (2007), pp. 250–262.

[15] K. A. GALLIVAN, C. QI, AND P.-A. ABSIL, *A Riemannian Dennis–Moré condition*, tech. report, Florida State University, 2012.

[16] N. J. HIGHAM, *Functions of Matrices*, SIAM, Philadelphia, Pennsylvania, USA, 2008.

[17] J. LAWSON AND Y. LIM, *Monotonic properties of the least squares mean*, Mathematische Annalen, 351 (2011), pp. 267–279.

[18] J. M. LEE, *Manifolds and Differential Geometry*, Graduate Studies in Mathematics, American Mathematical Society, 2009.

[19] M. MOAKHER, *A differential geometric approach to the geometric mean of symmetric positive-definite matrices*, SIAM Journal on Matrix Analysis and Applications, 26 (2005), pp. 735–747.

[20] M. MOAKHER, *On the averaging of symmetric positive-definite tensors*, Journal of Elasticity, 82 (2006), pp. 273–296.

[21] K. NAKAMURA, *Geometric means of positive operators*, Kyunpook Mathematical Journal, 49 (2009), pp. 167–181.

[22] J. NOCEDAL AND S. J. WRIGHT, *Numerical optimization*, Springer–Verlag, New York, 1999.

[23] M. PALFIA, *The Riemannian barycenter computation and means of several matrices*, International Journal of Computational and Mathematical Sciences, 3 (2009), pp. 128–133.

[24] F. POLONI, *Constructing matrix geometric means*, Electronic Journal of Linear Algebra, (2010), pp. 419–435.

[25] C. QI, *Numerical optimization methods on Riemannian manifolds*, PhD thesis, Florida State University, College of arts and sciences, 2011.

[26] Y. RATHI, A. TANNENBAUM, AND O. MICHAILOVICH, *Segmenting images on the tensor manifold*, in IEEE Conference on Computer Vision and Pattern Recognition, IEEE, 2007, pp. 1–8.

[27] Q. RENTMEESTER AND P.-A. ABSIL, *Algorithm comparison for Karcher mean computation of rotation matrices and diffusion tensors*, in European Signal Processing, 2011, EUSIPCO. 19th conference on, 2011.

[28] B. SAVAS AND L.-H. LIM, *Quasi-Newton Methods on Grassmannians and Multilinear Approximations of Tensors*, SIAM Journal on Scientific Computing, 32 (2010), pp. 3352–3393.

[29] B. VANDEREYCKEN, P.-A. ABSIL, AND S. VANDEWALLE, *Embedded geometry of the set of symmetric positive semidefinite matrices of fixed rank*, in Statistical Signal Processing, 2009. SSP'09. IEEE/SP 15th Workshop on, IEEE, IEEE, 2009, pp. 389–392.

[30] B. VANDEREYCKEN, P.-A. ABSIL, AND S. VANDEWALLE, *A Riemannian geometry with complete geodesics for the set of positive semidefinite matrices of fixed rank*, IMA Journal of Numerical Analysis, (2012). Accepted.

Recent publications :

MATHEMATICS INSTITUTE OF COMPUTATIONAL SCIENCE AND ENGINEERING
Section of Mathematics
Ecole Polytechnique Fédérale
CH-1015 Lausanne