A Riemannian Limited-Memory BFGS Algorithm for Computing the Matrix Geometric Mean

Xinru Yuan[1], Wen Huang[2], P.-A. Absil[2] and Kyle A. Gallivan[1]

[1]Florida State University

[2]Université Catholique de Louvain

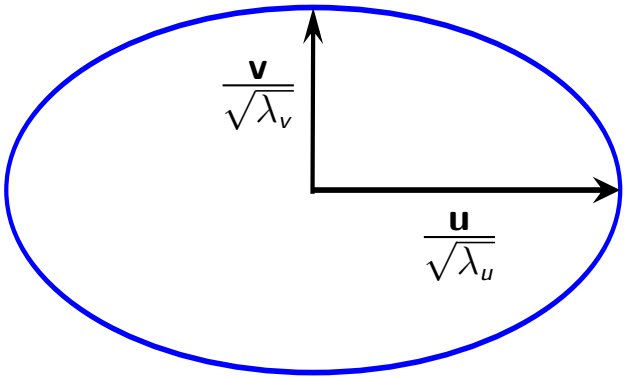
July 3, 2016

Symmetric Positive Definite (SPD) Matrix

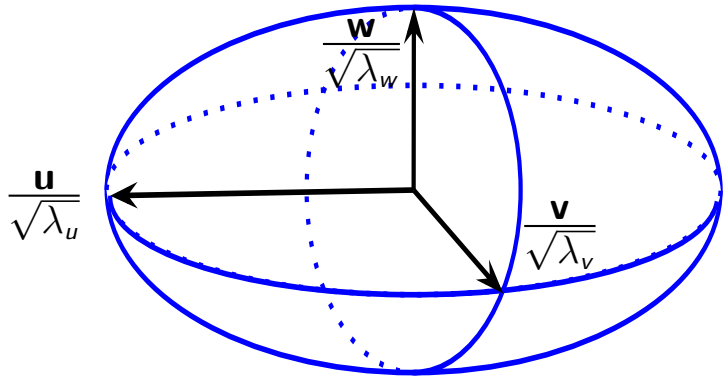
Definition

A symmetric matrix A is called positive definite if $x^T Ax > 0$ for all nonzero vector $x \in \mathbb{R}^n$.

2×2 SPD matrix

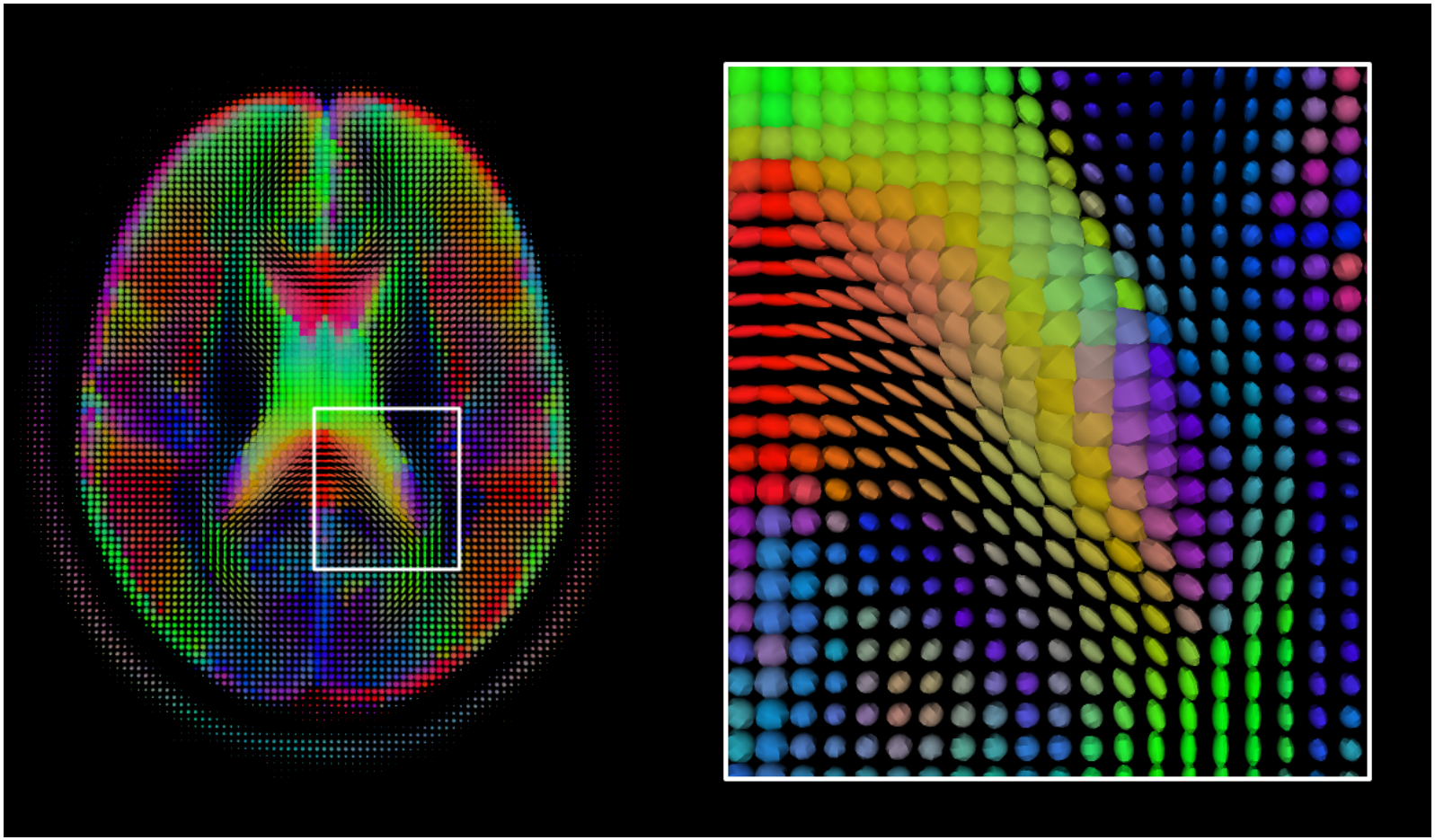


3×3 SPD matrix



Possible Applications of SPD Matrices

- Diffusion tensors in medical imaging



Motivation of Averaging SPD Matrices

- Subtask in interpolation schemes
  - Example: retrieve a tensor field between measured tensor values at sparse points [PFA06][1]

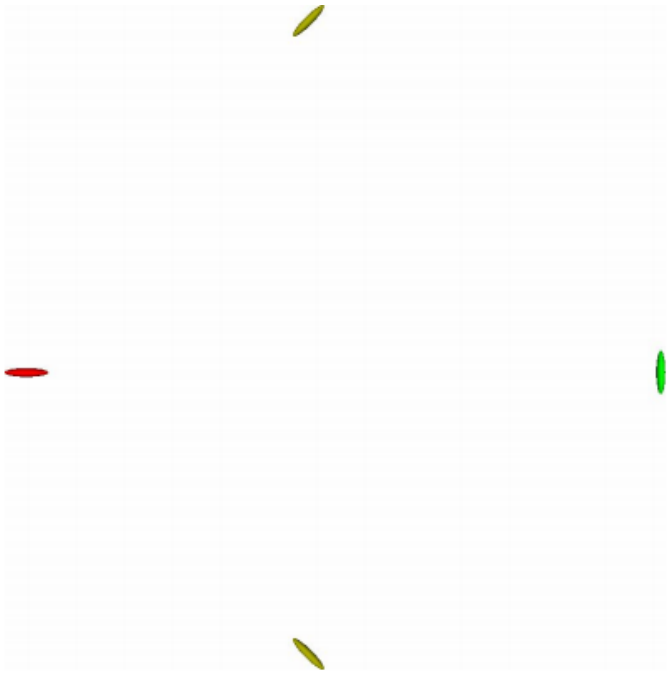


Figure : Initial tensor measurements

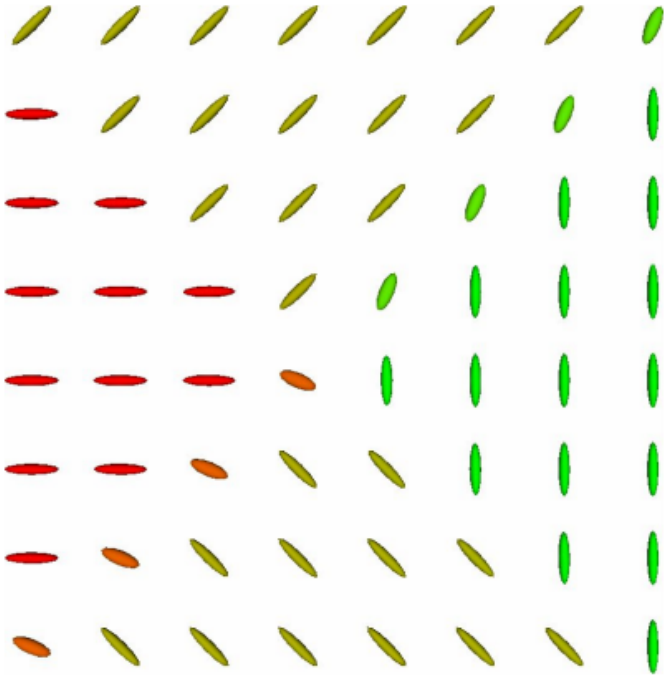


Figure : Retrieved tensor field after a certain interpolation scheme

[1] X. Pennec, P. Fillard and N. Ayache, *A Riemannian framework for tensor computing*, International Journal of Computer Vision, 2006

Averaging Schemes: from Scalars to Matrices

Let $A_1, \ldots, A_K$ be SPD matrices.

- Generalized arithmetic mean: $\frac{1}{K} \sum_{i=1}^{K} A_i$

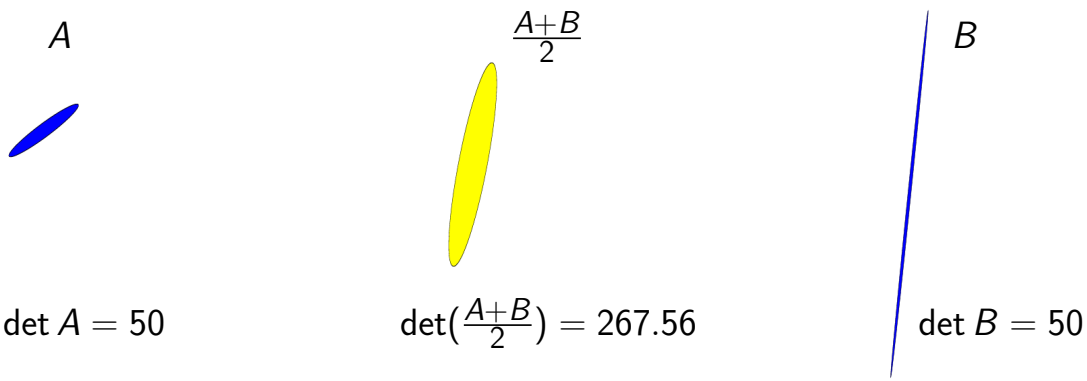
  $\rightarrow$ Not appropriate in many practical applications

Averaging Schemes: from Scalars to Matrices

Let $A_1, \ldots, A_K$ be SPD matrices.

- Generalized arithmetic mean: $\frac{1}{K} \sum_{i=1}^{K} A_i$

  $\rightarrow$ Not appropriate in many practical applications

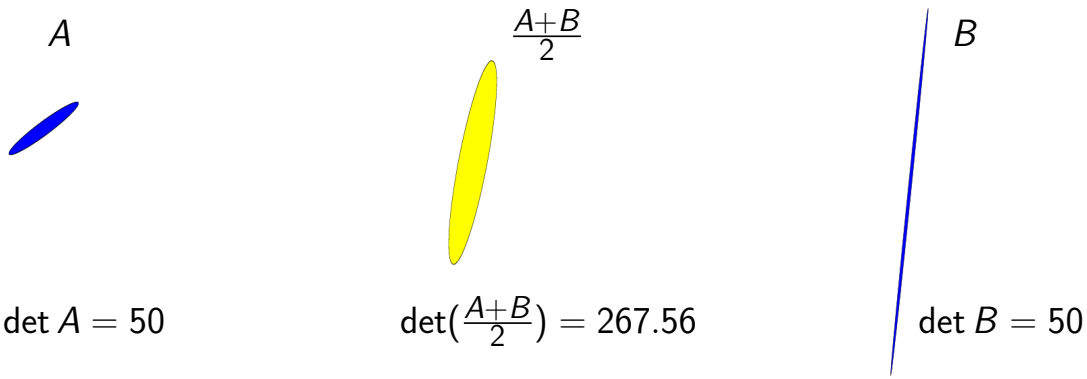


Averaging Schemes: from Scalars to Matrices

Let $A_1, \ldots, A_K$ be SPD matrices.

- Generalized arithmetic mean: $\frac{1}{K} \sum_{i=1}^{K} A_i$

  $\rightarrow$ Not appropriate in many practical applications



- Generalized geometric mean: $(A_1 \cdots A_K)^{1/K}$

  $\rightarrow$ Not appropriate due to non-commutativity

  $\rightarrow$ How to define a matrix geometric mean?

Desired Properties of a Matrix Geometric Mean

The desired properties are given in the ALM list[2], some of which are:

- Invariance under permutation:
  $G(A_{\pi(1)}, \ldots, A_{\pi(K)}) = G(A_1, \ldots, A_K)$ with π a permutation of $(1, \ldots, K)$

[2] T. Ando, C.-K. Li, and R. Mathias, *Geometric means*, Linear Algebra and Its Applications, 385:305-334, 2004

Desired Properties of a Matrix Geometric Mean

The desired properties are given in the ALM list[2], some of which are:

- Invariance under permutation:
  $G(A_{\pi(1)}, \ldots, A_{\pi(K)}) = G(A_1, \ldots, A_K)$ with π a permutation of $(1, \ldots, K)$
- Consistency with scalars:
  if $A_1, \ldots, A_K$ commute, then $G(A_1, \ldots, A_K) = (A_1, \ldots, A_K)^{1/K}$

[2] T. Ando, C.-K. Li, and R. Mathias, *Geometric means*, Linear Algebra and Its Applications, 385:305-334, 2004

Desired Properties of a Matrix Geometric Mean

The desired properties are given in the ALM list[2], some of which are:

- Invariance under permutation:
  $G(A_{\pi(1)}, \ldots, A_{\pi(K)}) = G(A_1, \ldots, A_K)$ with π a permutation of $(1, \ldots, K)$
- Consistency with scalars:
  if $A_1, \ldots, A_K$ commute, then $G(A_1, \ldots, A_K) = (A_1, \ldots, A_K)^{1/K}$
- Invariance under inversion:
  $G(A_1, \ldots, A_K)^{-1} = G(A_1^{-1}, \ldots, A_K^{-1})$

[2] T. Ando, C.-K. Li, and R. Mathias, *Geometric means*, Linear Algebra and Its Applications, 385:305-334, 2004

Desired Properties of a Matrix Geometric Mean

The desired properties are given in the ALM list[2], some of which are:

- Invariance under permutation:
  $G(A_{\pi(1)}, \ldots, A_{\pi(K)}) = G(A_1, \ldots, A_K)$ with π a permutation of $(1, \ldots, K)$
- Consistency with scalars:
  if $A_1, \ldots, A_K$ commute, then $G(A_1, \ldots, A_K) = (A_1, \ldots, A_K)^{1/K}$
- Invariance under inversion:
  $G(A_1, \ldots, A_K)^{-1} = G(A_1^{-1}, \ldots, A_K^{-1})$
- Determinant equality:
  $\det(G(A_1, \ldots, A_K)) = (\det(A_1) \cdots \det(A_K))^{1/K}$

[2] T. Ando, C.-K. Li, and R. Mathias, *Geometric means*, Linear Algebra and Its Applications, 385:305-334, 2004

Geometric Mean of SPD Matrices

A well-known mean on the manifold of SPD matrices is the **Karcher mean** [Kar77]:

$$G(A_1, \ldots, G_K) = \arg\min_{X \in \mathcal{S}^n_{++}} \frac{1}{2K} \sum_{i=1}^{K} \mathrm{dist}^2(X, A_i),$$

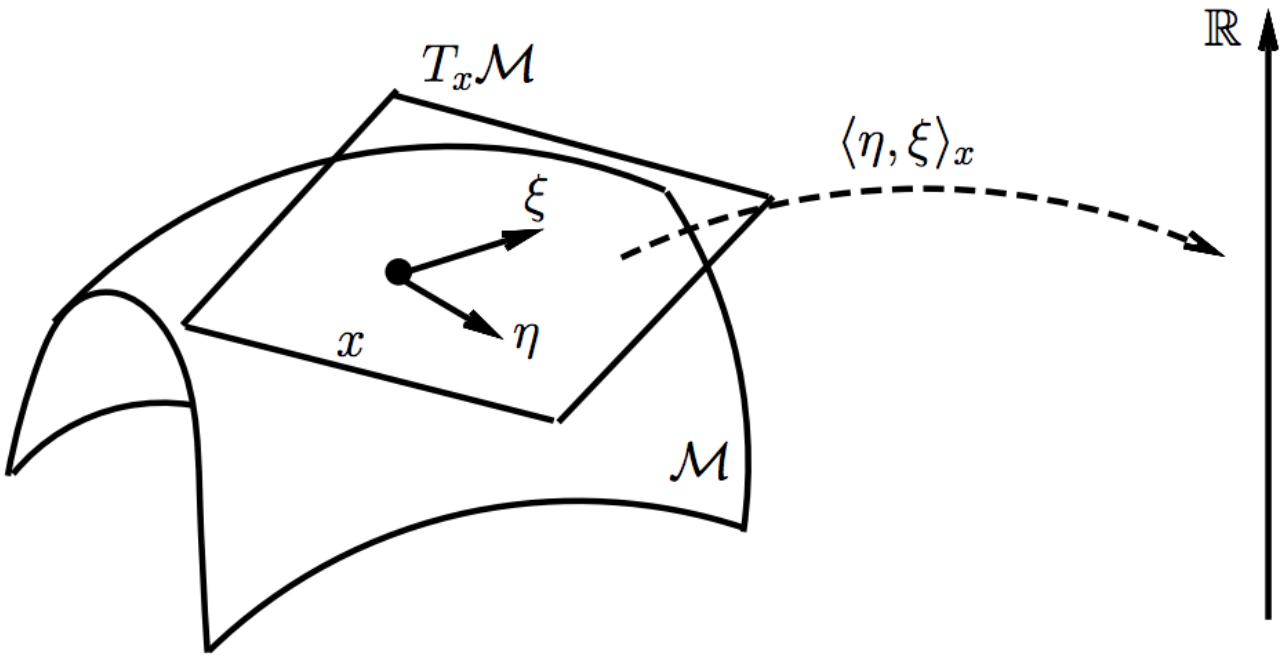
where $\mathrm{dist}(X, Y) = \| \log(X^{-1/2} Y X^{-1/2}) \|_F$ is the distance under the Riemannian metric

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

It has been shown recently [LL11] that the Karcher mean satisfies all the geometric properties in the ALM list.

Riemannian Manifolds

A Riemannian manifold $\mathcal{M}$ is a smooth set with a smoothly-varying inner product on the tangent spaces.



Manifold of SPD Matrices

- Let $\mathcal{S}_{++}^n$ be the manifold of SPD matrices
- Tangent space at $X \in \mathcal{S}_{++}^n$:

$$\mathrm{T}_X \mathcal{S}_{++}^n = \{S \in \mathbb{R}^{n \times n} : S = S^T\}$$

- Dimension: $d = n(n+1)/2$
- Riemannian metric:

$$g(\eta_X, \xi_X) = \mathrm{trace}(\eta_X X^{-1} \xi_X X^{-1})$$

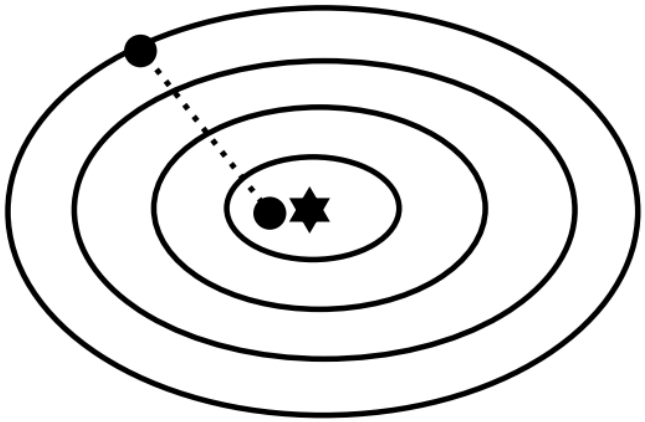
Algorithms

$$G(A_1, \ldots, G_K) = \arg\min_{X \in \mathcal{S}_{++}^n} \frac{1}{2K} \sum_{i=1}^{K} \mathrm{dist}^2(X, A_i),$$

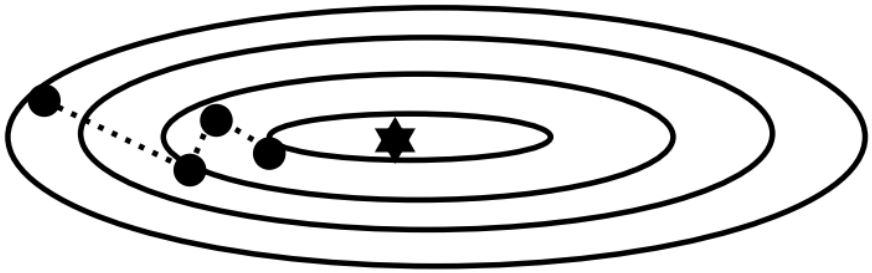
- Riemannian steepest descent [RA11]
- Riemannian steepest descent, conjugate gradient, BFGS, and trust region Newton methods [JVV12]
- Richardson-like iteration [BI13]
- Limited-memory Riemannian BFGS method [YHAG16]

Karcher Mean of SPD Matrices

Hemstitching phenomenon
for steepest descent



well-conditioned Hessian



ill-conditioned Hessian

- Small condition number $\Rightarrow$ fast convergence
- Large condition number $\Rightarrow$ slow convergence

Karcher Mean of SPD Matrices

Conditioning of the Riemannian Hessian of F at the minimizer:

- For the cost function $F(X) = \frac{1}{2K}\sum_{i=1}^{K} \mathrm{dist}^2(A_i, X)$, we have

$$1 \leq \frac{\mathrm{Hess}\, F(X)[\Delta X, \Delta X]}{\|\Delta X\|^2} \leq 1 + \frac{\ln(\max \kappa_i)}{2},$$

  where κ_i is the condition number of A_i.

- If $\max \kappa_i = 10^{10}$, then $1 + \frac{\ln(\max \kappa_i)}{2} \approx 12.51$.

Optimization Algorithm: from Euclidean to Riemannian

- Update formula:

$$x_{k+1} = x_k + \alpha_k \eta_k,$$

  where η_k is the search direction and α_k is the stepsize.

- Search direction:

$$\eta_k = -B_k^{-1} \operatorname{grad} f(x_k).$$

Optimization Algorithm: from Euclidean to Riemannian

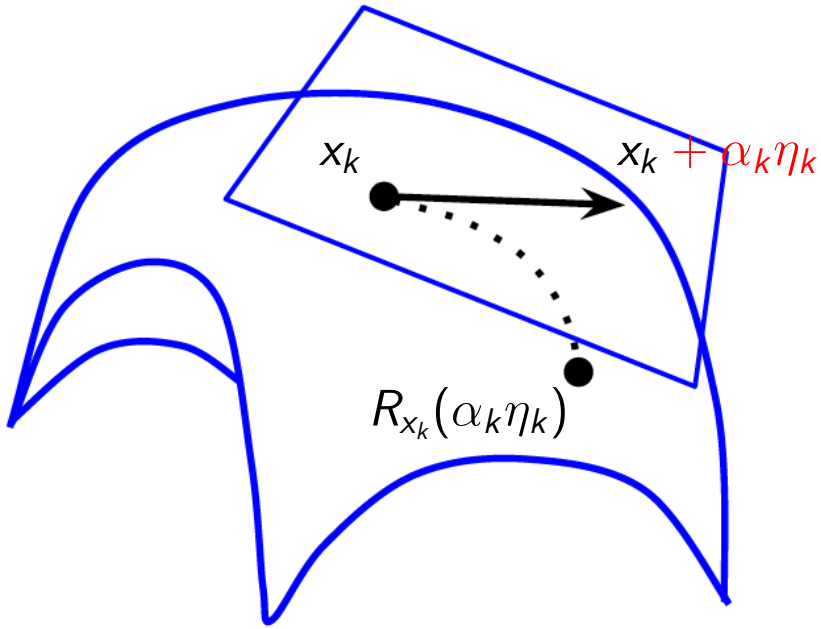
- Update formula:

$$x_{k+1} = x_k + \alpha_k \eta_k,$$

where η_k is the search direction and α_k is the stepsize.

- Search direction:

$$\eta_k = -B_k^{-1} \operatorname{grad} f(x_k).$$



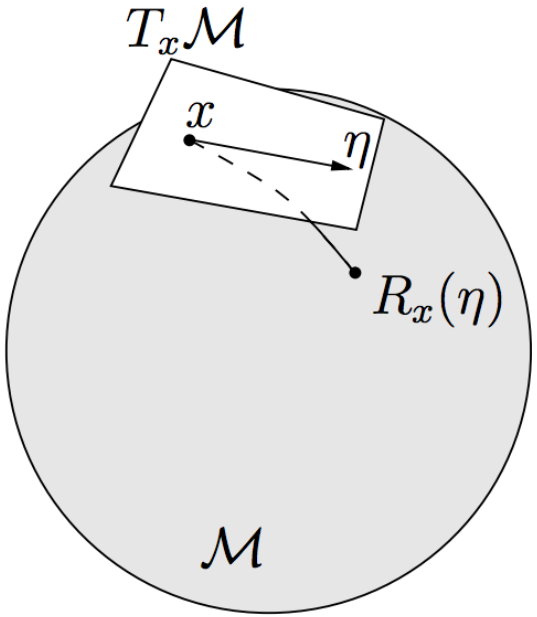
Optimization on a Manifold

Retraction

Definition

A retraction is a mapping R from $T\mathcal{M}$ to $\mathcal{M}$ satisfying the following:

- R is continuously differentiable
- $R_x(0) = x$
- $\mathrm{D}R_x(0)(\eta) = \eta$



Euclidean	Riemannian
$x_{k+1} = x_k + \alpha_k \eta_k$	$x_{k+1} = R_{x_k}(\alpha_k \eta_k)$

Riemannian Optimization Algorithm

Euclidean	Riemannian
$x_{k+1} = x_k + \alpha_k \eta_k$	$x_{k+1} = R_{x_k}(\alpha_k \eta_k)$

- Steepest Descent:

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

- Newton's Method:

$$\eta_k = -\operatorname{Hess} f(x_k)^{-1} \operatorname{grad} f(x_k)$$

- Quasi-Newton Method:

$$\eta_k = -B_k^{-1} \operatorname{grad} f(x_k)$$

Riemannian Gradient and Hessian

- Riemannian gradient:

$$\operatorname{grad} F(X) = \frac{1}{K}\sum_{i=1}^{K} A_i^{1/2}\log(A_i^{-1/2}XA_i^{-1/2})A_i^{-1/2}X^{1/2}$$

- Riemannian Hessian:

$$\begin{aligned}\operatorname{Hess} F(X)[\xi_X] =& \frac{1}{2K}\sum_{i=1}^{K}\xi_X\log(A_i^{-1}X) - \frac{1}{2K}\sum_{i=1}^{K}\log(XA_i^{-1})\xi_X \\ &+ \frac{1}{K}\sum_{i=1}^{K} X\mathrm{D}(\log)(A_i^{-1}X)[A_i^{-1}\xi_X]\end{aligned}$$

Richardson-like Iteration [BI13]

- Update formula:

$$X_{k+1} = X_k - \alpha X_k^{1/2} \sum_{i=1}^{K} \log(X_k^{1/2} A_i^{-1} X_k^{1/2}) X_k^{1/2}$$

- Stepsize:

$$\alpha = 2/\sum_{i=1}^{K} \frac{\kappa_i + 1}{\kappa_i - 1} \log \kappa_i,$$

  where κ_i is the condition number of matrix $X_k^{1/2} A_i^{-1} X_k^{1/2}$

- Local linear convergence is proved

Richardson-like Iteration, Ctd

- The Richardson-like iteration can be considered from the view of Riemannian optimization algorithm

- $X_{k+1} = R_{X_k}(\alpha \eta_k)$
  - Search direction: $\eta_k = -\operatorname{grad} F(X_k)$
  - Choice of retraction: $R_X(\eta) = X + \eta$
  - Stepsize α satisfies the Wolfe condition

BFGS Quasi-Newton Algorithm: from Euclidean to Riemannian

- Search direction:

$$\eta_k = -B_k^{-1} \operatorname{grad} f(x_k)$$

- B_k update:

$$B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T s_k},$$

where $s_k = x_{k+1} - x_k$, and $y_k = \operatorname{grad} f(x_{k+1}) - \operatorname{grad} f(x_k)$.

BFGS Quasi-Newton Algorithm: from Euclidean to Riemannian

- Search direction:

$$\eta_k = -B_k^{-1} \operatorname{grad} f(x_k)$$

- B_k update:

$$B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T s_k},$$

where $s_k = x_{k+1} - x_k$, and $y_k = \operatorname{grad} f(x_{k+1}) - \operatorname{grad} f(x_k)$.

On a manifold:

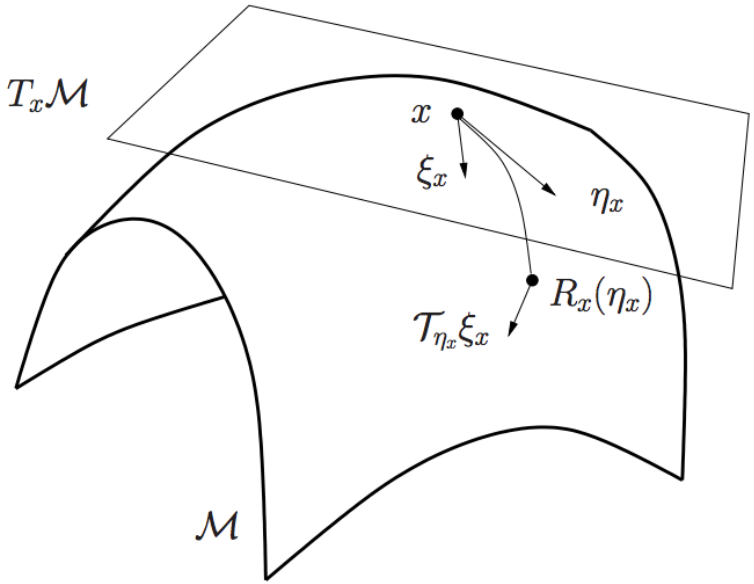
replaced by $R_{x_k}^{-1}(x_{k+1})$

on different tangent spaces

Vector Transport

Vector Transport

- Transport a tangent vector from one tangent space to another
- $\mathcal{T}_{\eta_x}\xi_x$, denotes transport of ξ_x to tangent space of $R_x(\eta_x)$



Riemannian BFGS (RBFGS) Algorithm

- Update formula:

$$x_{k+1} = R_{x_k}(\alpha_k \eta_k) \text{ with } \eta_k = -\mathcal{B}_k^{-1} \operatorname{grad} f(x_k)$$

- $\mathcal{B}_k$ update [HGA15]:

$$\mathcal{B}_{k+1} = \tilde{\mathcal{B}}_k - \frac{\tilde{\mathcal{B}}_k s_k (\tilde{\mathcal{B}}_k s_k)^\flat}{(\tilde{\mathcal{B}}_k s_k)^\flat s_k} + \frac{y_k y_k^\flat}{y_k^\flat s_k},$$

where $s_k = \mathcal{T}_{\alpha_k \eta_k} \alpha_k \eta_k$, $y_k = \beta_k^{-1} \operatorname{grad} f(x_{k+1}) - \mathcal{T}_{\alpha_k \eta_k} \operatorname{grad} f(x_k)$, and $\tilde{\mathcal{B}}_k = \mathcal{T}_{\alpha_k \eta_k} \circ \mathcal{B}_k \circ \mathcal{T}_{\alpha_k \eta_k}^{-1}$.

Riemannian BFGS (RBFGS) Algorithm

- Update formula:

$$x_{k+1} = R_{x_k}(\alpha_k \eta_k) \text{ with } \eta_k = -\mathcal{B}_k^{-1} \operatorname{grad} f(x_k)$$

- $\mathcal{B}_k$ update [HGA15]:

$$\mathcal{B}_{k+1} = \tilde{\mathcal{B}}_k - \frac{\tilde{\mathcal{B}}_k s_k (\tilde{\mathcal{B}}_k s_k)^\flat}{(\tilde{\mathcal{B}}_k s_k)^\flat s_k} + \frac{y_k y_k^\flat}{y_k^\flat s_k},$$

  where $s_k = \mathcal{T}_{\alpha_k \eta_k} \alpha_k \eta_k$, $y_k = \beta_k^{-1} \operatorname{grad} f(x_{k+1}) - \mathcal{T}_{\alpha_k \eta_k} \operatorname{grad} f(x_k)$, and $\tilde{\mathcal{B}}_k = \mathcal{T}_{\alpha_k \eta_k} \circ \mathcal{B}_k \circ \mathcal{T}_{\alpha_k \eta_k}^{-1}$.

- Stores and transports $\mathcal{B}_k^{-1}$ as a dense matrix

- Requires excessive computation time and storage space for large-scale problem

Limited-memory RBFGS (LRBFGS)

Riemannian BFGS:

- $\mathcal{B}_{k+1} = \tilde{\mathcal{B}}_k - \frac{\tilde{\mathcal{B}}_k s_k (\tilde{\mathcal{B}}_k s_k)^\flat}{(\tilde{\mathcal{B}}_k s_k)^\flat s_k} + \frac{y_k y_k^\flat}{y_k^\flat s_k}$,
  where $s_k = \mathcal{T}_{\alpha_k \eta_k} \alpha_k \eta_k$, $y_k = \beta_k^{-1} \operatorname{grad} f(x_{k+1}) - \mathcal{T}_{\alpha_k \eta_k} \operatorname{grad} f(x_k)$,
  and $\tilde{\mathcal{B}}_k = \mathcal{T}_{\alpha_k \eta_k} \circ \mathcal{B}_k \circ \mathcal{T}^{-1}_{\alpha_k \eta_k}$

Limited-memory Riemannian BFGS:

- Stores only the m most recent s_k and y_k
- Transports those vectors to the new tangent space rather than the entire matrix $\mathcal{B}_k^{-1}$
- Computational and storage complexity depends upon m

Limited-memory RBFGS (LRBFGS), Ctd

- $\mathcal{B}_k^{-1}$ update [HGA15]:

$$\begin{aligned}\mathcal{B}_k^{-1} &= \tilde{\mathcal{V}}_k^\flat \tilde{\mathcal{V}}_{k-1}^\flat \cdots \tilde{\mathcal{V}}_{k-m}^\flat \tilde{\mathcal{H}}_{k+1}^0 \tilde{\mathcal{V}}_{k-m} \cdots \tilde{\mathcal{V}}_{k-1} \tilde{\mathcal{V}}_k \\ &\quad + \rho_{k-m} \tilde{\mathcal{V}}_k^\flat \tilde{\mathcal{V}}_{k-1}^\flat \cdots \tilde{\mathcal{V}}_{k-m+1}^\flat s_{k-m}^{(k+1)} s_{k-m}^{(k+1)^\flat} \tilde{\mathcal{V}}_{k-m+1} \cdots \tilde{\mathcal{V}}_{k-1} \tilde{\mathcal{V}}_k \\ &\quad + \cdots + \rho_k s_k^{(k+1)} s_k^{(k+1)^\flat},\end{aligned}$$

  where $\tilde{\mathcal{V}}_i = \mathrm{id} - \rho_i y_i^{(k+1)} s_i^{(k+1)^\flat}$, $\rho_i = \frac{1}{g(y_i, s_i)}$, $\tilde{\mathcal{H}}_{k+1}^0 = \frac{g(s_k, y_k)}{g(y_k, y_k)}\,\mathrm{id}$, and $s_i^{(k+1)}$ represents a tangent vector in $\mathrm{T}_{x_{k+1}}\mathcal{M}$ given by transporting s_i, and likewise for $y_i^{(k+1)}$

- Avoids expensive matrix operations

- Requires less memory

Complexity Comparison

RBFGS		Limited-memory RBFGS	
Action	Complexity	Action	Complexity
get $\tilde{\mathcal{B}}_{k+1}^{-1}$ from $\mathcal{B}_k^{-1}$	$O(d^2)$	-	-
$\mathcal{B}^{-1}\eta$	$O(d^2)$	$\mathcal{B}^{-1}\eta$	$O(md)$

- d denotes the dimension of SPD manifold

Algorithm Comparison

Not sure where to put this table

Algorithm	Convergence
Riemannian steepest descent	global, linear
Riemannian BFGS	global(convex), superlinear
Riemannian limited-memory BFGS	global(convex), faster than linear
Riemannian Newton's method	global, quadratic

Outline

- Background of the SPD Karcher mean computation
- A brief description of the limited-memory RBFGS (LRBFGS) method
- Apply the LRBFGS to the SPD Karcher mean computation
  - Practical implementation
  - Experiments on various data sets

Implementations

- Cost function:

$$F(X) = \frac{1}{2K}\sum_{i=1}^{K} \mathrm{dist}^2(A_i, X)$$
$$= \frac{1}{2K}\sum_{i=1}^{K} \| \log(A_i^{-1/2} X A_i^{-1/2}) \|_F^2$$

- Riemannian gradient:

$$\mathrm{grad}\, F(X) = \frac{1}{K}\sum_{i=1}^{K} A_i^{1/2} \log(A_i^{-1/2} X A_i^{-1/2}) A_i^{-1/2} X^{1/2}$$

- The dominant computation time is on the function evaluation

Implementations

- Efficient representations of tangent vectors, see [YHAG16]
- Retraction
  - Exponential mapping: $\mathrm{Exp}_X(\xi_X) = X^{1/2}\exp(X^{-1/2}\xi_X X^{-1/2})X^{1/2}$
- Vector transport
  - Parallel translation: $\mathcal{T}_{p\eta_X}(\xi_X) = Q\xi_X Q^T$,

    where $Q = X^{1/2}\exp(\frac{X^{-1/2}\eta_X X^{-1/2}}{2})X^{-1/2}$

Implementations

- Efficient numerical representations of tangent vectors
- Retraction
  - Exponential mapping: $\mathrm{Exp}_X(\xi_X) = X^{1/2}\exp(X^{-1/2}\xi_X X^{-1/2})X^{1/2}$
  - Second order retraction [JVV12]: $R_X(\xi_X) = X + \xi_X + \frac{1}{2}\xi_X X^{-1}\xi_X$
- Vector transport
  - Parallel translation: $\mathcal{T}_{p\eta_X}(\xi_X) = Q\xi_X Q^T$,

    where $Q = X^{1/2}\exp(\frac{X^{-1/2}\eta_X X^{-1/2}}{2})X^{-1/2}$

Implementations

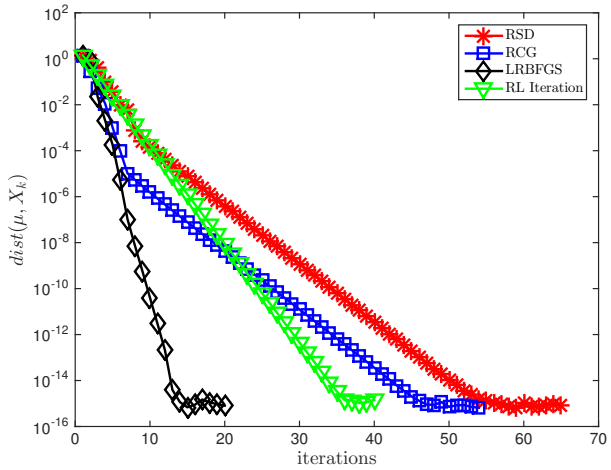
- Efficient numerical representations of tangent vectors
- Retraction
  - Exponential mapping: $\mathrm{Exp}_X(\xi_X) = X^{1/2}\exp(X^{-1/2}\xi_X X^{-1/2})X^{1/2}$
  - Second order retraction: $R_X(\xi_X) = X + \xi_X + \frac{1}{2}\xi_X X^{-1}\xi_X$
- Vector transport
  - Parallel translation: $\mathcal{T}_{p\eta_X}(\xi_X) = Q\xi_X Q^T$,

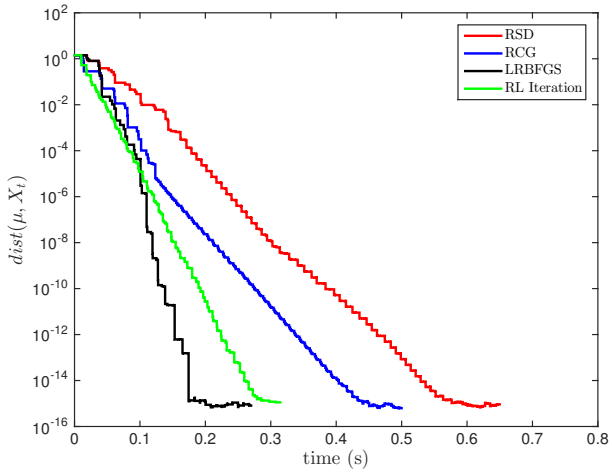
    where $Q = X^{1/2}\exp(\frac{X^{-1/2}\eta_X X^{-1/2}}{2})X^{-1/2}$
  - Vector transport by parallelization[3]: essentially an identity

[3] W. Huang, P.-A. Absil, and K. A. Gallivan, *A Riemannian symmetric rank-one trust-region method*, Mathematical Programming, 2014

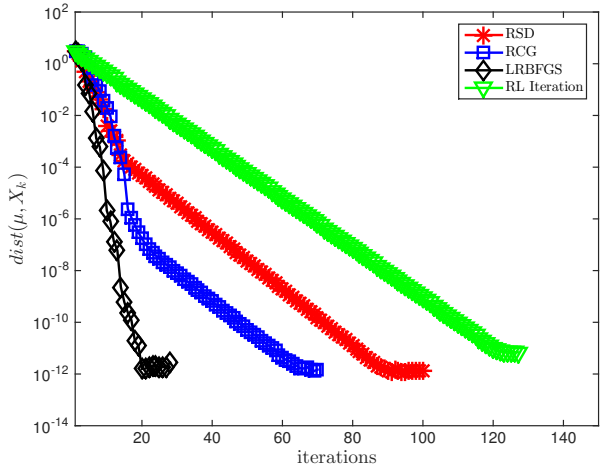
Numerical Results I: $K = 100$, 3×3, $d = 6$, $m = 2$

- $1 \leq \kappa(A_i) \leq 200$





- $10^3 \leq \kappa(A_i) \leq 2 \cdot 10^6$



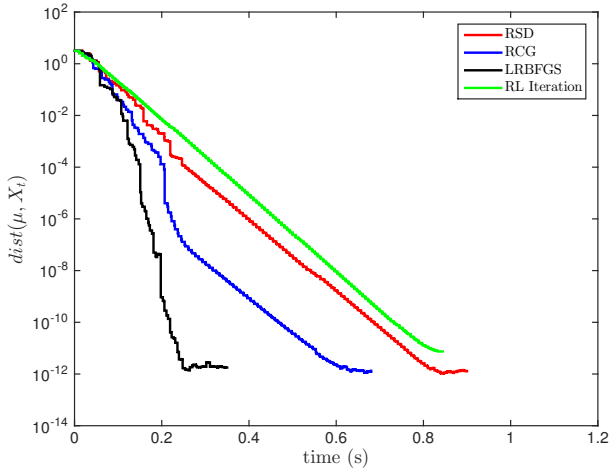
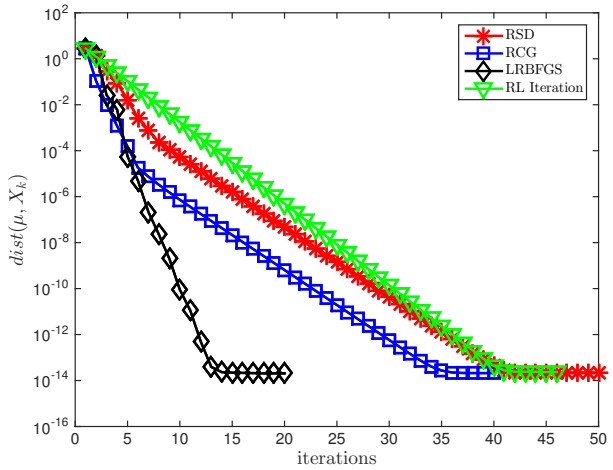
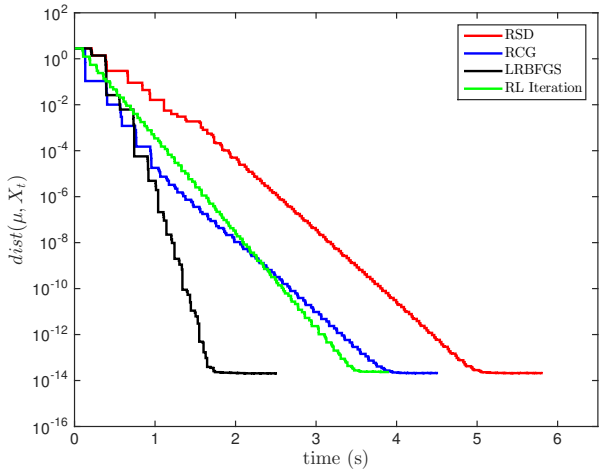


Figure : Evolution of averaged distance between current iterate and the exact Karcher mean with respect to time and iterations

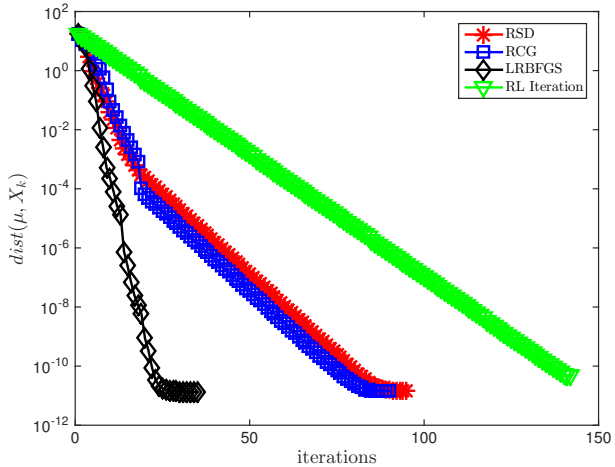
Numerical Results II: $K = 30$, 100×100, $d = 5050$, $m = 2$

- $1 \leq \kappa(A_i) \leq 20$





- $10^4 \leq \kappa(A_i) \leq 2 \cdot 10^6$



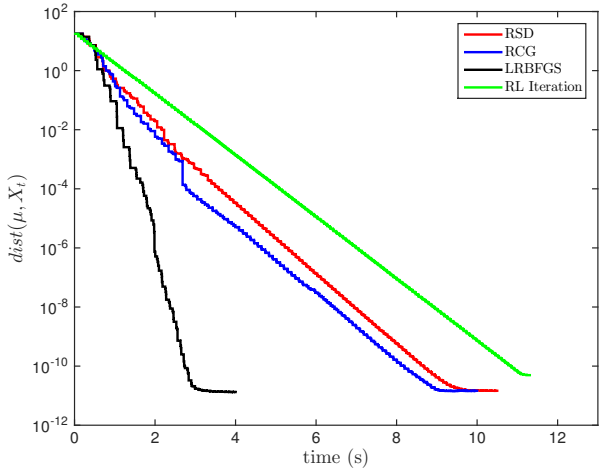


Figure : Evolution of averaged distance between current iterate and the exact Karcher mean with respect to time and iterations

Summary

- Analyze the conditioning of the Hessian of the Karcher cost function
- Apply a Riemannian version of limited-memory BFGS method to computing the SPD Karcher mean
- Present efficient implementations
- Compare with the state-of-the-art methods
- The proposed LRBFGS method is seen to be the method of choice for computing the SPD Karcher mean when the data matrices increase in size or number.

Implementation in Matlab

$$
\begin{aligned}
F(X) &= \frac{1}{2K}\sum_{i=1}^{K} \| \log(A_i^{-1/2} X A_i^{-1/2}) \|_F^2 \\
&= \frac{1}{2K}\sum_{i=1}^{K} \| \log(A_i^{-1/2} L L^T A_i^{-1/2}) \|_F^2 \\
&= \frac{1}{2K}\sum_{i=1}^{K} \| \log[(A_i^{-1/2} L)(A_i^{-1/2} L)^T] \|_F^2
\end{aligned}
$$

Let $S = A_i^{-1/2} L$, and $SS^T = QUQ^T$, where $QQ^T = I$, and U is an upper triangle matrix whose diagonal elements are the eigenvalues. Then

$$
\begin{aligned}
F(X) &= \frac{1}{2K}\sum_{i=1}^{K} \| \log(SS^T) \|_F^2 \\
&= \frac{1}{2K}\sum_{i=1}^{K} \| Q \log(\mathrm{diag}(U)) Q^T \|_F^2
\end{aligned}
$$

References I


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.


W. Huang, K. 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.


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.


H. Karcher.
Riemannian center of mass and mollifier smoothing.
Communications on Pure and Applied Mathematics, 1977.

References II


Jimmie Lawson and Yongdo Lim.
Monotonic properties of the least squares mean.
Mathematische Annalen, 351(2):267–279, 2011.


Xavier Pennec, Pierre Fillard, and Nicholas Ayache.
A riemannian framework for tensor computing.
International Journal of Computer Vision, 66(1):41–66, 2006.


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.


Xinru Yuan, Wen Huang, P.-A. Absil, and Kyle A. Gallivan.
A Riemannian limited-memory BFGS algorithm for computing the matrix geometric mean.
In *ICCS 2016*, 2016.