Package 'RiemBase'

September 23, 2025

Type Package

Title Functions and C++ Header Files for Computation on Manifolds

Version 0.2.6

Description We provide a number of algorithms to estimate fundamental statistics including Fréchet mean and geometric median for manifold-valued data. Also, C++ header files are contained that implement elementary operations on manifolds such as Sphere, Grassmann, and others. See Bhattacharya and Bhattacharya (2012) <doi:10.1017/CBO9781139094764> if you are interested in statistics on manifolds, and Absil et al (2007, ISBN:9780691132983) on computational aspects of optimization on matrix manifolds.

Encoding UTF-8

Depends R (>= 3.0.0)

License MIT + file LICENSE

Imports Rcpp, utils, Rdpack, parallel, stats, pracma

LinkingTo Rcpp, RcppArmadillo

RoxygenNote 7.3.2

RdMacros Rdpack

URL https://github.com/kisungyou/RiemBase

BugReports https://github.com/kisungyou/RiemBase/issues

NeedsCompilation yes

Author Kisung You [aut, cre] (ORCID: <https://orcid.org/0000-0002-8584-459X>)

Maintainer Kisung You <kisung.you@outlook.com>

Repository CRAN

Date/Publication 2025-09-23 07:40:07 UTC

Contents

`rbase.curvedist` *Distance between Two Curves with Finite Difference Approximation*

Description

Suppose we have to two curves $f, g : I \subset \mathbf{R} \to \mathcal{M}$ evaluated at finite locations $t_0 \leq \ldots \leq t_N$, `rbase.curvedist` computes distance between two curves f and g using finite difference approximation with trapezoidal rule. In order to induce no interpolation, two curves should be of same length.

Usage

```
rbase.curvedist(curve1, curve2, t = NULL, type = c("intrinsic", "extrinsic"))
```

Arguments

`curve1`	a S3 object of `riemdata` class, whose `$data` element is of length N.
`curve2`	a S3 object of `riemdata` class, whose `$data` element is of length N.
`t`	a length-N vector of locations. If `NULL` is given, it uses a equidistanct sequence from 1 to N.
`type`	type of Riemannian distance (`"intrinsic"` or `"extrnisic"`).

Value

computed distance.

Examples

```
## Not run:
### Generate two sets of 10 2-frames in R^4 : as grassmann points
ndata = 10
data1 = array(0,c(4,2,ndata))
data2 = array(0,c(4,2,ndata))
for (i in 1:ndata){
  tgt = matrix(rnorm(4*4),nrow=4)
  data1[,,i] = qr.Q(qr(tgt))[,1:2]
}
for (i in 1:ndata){
  tgt = matrix(rnorm(4*5, sd=2),nrow=4)
  data2[,,i] = qr.Q(qr(tgt))[,1:2]
}
```

```
gdata1 = riemfactory(data1, name="grassmann") # wrap as 'riemdata' class.
gdata2 = riemfactory(data2, name="grassmann")

rbase.curvedist(gdata1, gdata2)

## End(Not run)
```

`rbase.mean` *Fréchet Mean of Manifold-valued Data*

Description

For manifold-valued data, Fréchet mean is the solution of following cost function,

$$\min_x \sum_{i=1}^{n} \rho^2(x, x_i), \quad x \in \mathcal{M}$$

for a given data $\{x_i\}_{i=1}^n$ and $\rho(x, y)$ is the geodesic distance between two points on manifold $\mathcal{M}$. It uses a gradient descent method with a backtracking search rule for updating.

Usage

```
rbase.mean(input, maxiter = 496, eps = 1e-06, parallel = FALSE)
```

Arguments

`input`	a S3 object of `riemdata` class. See `riemfactory` for more details.
`maxiter`	maximum number of iterations for gradient descent algorithm.
`eps`	stopping criterion for the norm of gradient.
`parallel`	a flag for enabling parallel computation.

Value

a named list containing

x an estimate Fréchet mean.

iteration number of iterations until convergence.

Author(s)

Kisung You

References

Karcher H (1977). "Riemannian center of mass and mollifier smoothing." *Communications on Pure and Applied Mathematics*, **30**(5), 509–541. ISSN 00103640, 10970312.

Kendall WS (1990). "Probability, Convexity, and Harmonic Maps with Small Image I: Uniqueness and Fine Existence." *Proceedings of the London Mathematical Society*, **s3-61**(2), 371–406. ISSN 00246115.

Afsari B, Tron R, Vidal R (2013). "On the Convergence of Gradient Descent for Finding the Riemannian Center of Mass." *SIAM Journal on Control and Optimization*, **51**(3), 2230–2260. ISSN 0363-0129, 1095-7138.

Examples

```
### Generate 100 data points on Sphere S^2 near (0,0,1).
ndata = 100
theta = seq(from=-0.99,to=0.99,length.out=ndata)*pi
tmpx  = cos(theta) + rnorm(ndata,sd=0.1)
tmpy  = sin(theta) + rnorm(ndata,sd=0.1)

### Wrap it as 'riemdata' class
data  = list()
for (i in 1:ndata){
  tgt = c(tmpx[i],tmpy[i],1)
  data[[i]] = tgt/sqrt(sum(tgt^2)) # project onto Sphere
}
data = riemfactory(data, name="sphere")

### Compute Fréchet Mean
out1 = rbase.mean(data)
out2 = rbase.mean(data,parallel=TRUE) # test parallel implementation
```

`rbase.median` *Geometric Median of Manifold-valued Data*

Description

For manifold-valued data, geometric median is the solution of following cost function,

$$\min_x \sum_{i=1}^{n} \rho(x, x_i) = \sum_{i=1}^{n} \| \log_x(x_i) \|, \quad x \in \mathcal{M}$$

for a given data $\{x_i\}_{i=1}^n$, $\rho(x, y)$ the geodesic distance between two points on manifold $\mathcal{M}$, and $\| \log_x(y) \|$ a logarithmic mapping onto the tangent space $T_x\mathcal{M}$. Weiszfeld's algorithms is employed.

Usage

```
rbase.median(input, maxiter = 496, eps = 1e-06, parallel = FALSE)
```

Arguments

input	a S3 object of riemdata class. See riemfactory for more details.
maxiter	maximum number of iterations for gradient descent algorithm.
eps	stopping criterion for the norm of gradient.
parallel	a flag for enabling parallel computation.

Value

a named list containing

x an estimate geometric median.

iteration number of iterations until convergence.

Author(s)

Kisung You

References

Fletcher PT, Venkatasubramanian S, Joshi S (2009). "The geometric median on Riemannian manifolds with application to robust atlas estimation." *NeuroImage*, **45**(1), S143–S152. ISSN 10538119.

Aftab K, Hartley R, Trumpf J (2015). "Generalized Weiszfeld Algorithms for Lq Optimization." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **37**(4), 728–745. ISSN 0162-8828, 2160-9292.

Examples

```
### Generate 100 data points on Sphere S^2 near (0,0,1).
ndata = 100
theta = seq(from=-0.99,to=0.99,length.out=ndata)*pi
tmpx  = cos(theta) + rnorm(ndata,sd=0.1)
tmpy  = sin(theta) + rnorm(ndata,sd=0.1)

### Wrap it as 'riemdata' class
data  = list()
for (i in 1:ndata){
  tgt = c(tmpx[i],tmpy[i],1)
  data[[i]] = tgt/sqrt(sum(tgt^2)) # project onto Sphere
}
data = riemfactory(data, name="sphere")

### Compute Geodesic Median
out1 = rbase.median(data)
out2 = rbase.median(data,parallel=TRUE) # test parallel implementation
```

`rbase.pdist` *Pairwise Geodesic Distances of a Data Set*

Description

Geodesic distance $\rho(x, y)$ is the length of (locally) shortest path connecting two points $x, y \in \mathcal{M}$. Some manifolds have closed-form expression, while others need numerical approximation.

Usage

```
rbase.pdist(input, parallel = FALSE)
```

Arguments

`input`	a S3 object of `riemdata` class, whose `$data` element is of length n. See `riemfactory` for more details.
`parallel`	a flag for enabling parallel computation.

Value

an $(n \times n)$ matrix of pairwise distances.

Examples

```
### Generate 10 2-frames in R^4
ndata = 10
data = array(0,c(4,2,ndata))
for (i in 1:ndata){
  tgt = matrix(rnorm(4*4),nrow=4)
  data[,,i] = qr.Q(qr(tgt))[,1:2]
}

## Compute Pairwise Distances as if for Grassmann and Stiefel Manifold
A = rbase.pdist(riemfactory(data,name="grassmann"))
B = rbase.pdist(riemfactory(data,name="stiefel"))

## Visual Comparison in Two Cases
opar = par(no.readonly=TRUE)
par(mfrow=c(1,2))
image(A, col=gray((0:100)/100), main="Grassmann")
image(B, col=gray((0:100)/100), main="Stiefel")
par(opar)
```

rbase.pdist2 *Pairwise Geodesic Distances Between Two Sets of Data*

Description

Unlike rbase.pdist, rbase.pdist2 takes two sets of data $X = \{x_i\}_{i=1}^m$ and $Y = \{y_j\}_{j=1}^m$ and compute mn number of pairwise distances for all i and j.

Usage

```
rbase.pdist2(input1, input2, parallel = FALSE)
```

Arguments

input1	a S3 object of riemdata class, whose $data element is of length m.
input2	a S3 object of riemdata class, whose $data element is of length n.
parallel	a flag for enabling parallel computation.

Value

an $(m \times n)$ matrix of pairwise distances.

Examples

```
### Generate 10 2-frames in R^4 : as grassmann points
ndata = 10
data = array(0,c(4,2,ndata))
for (i in 1:ndata){
  tgt = matrix(rnorm(4*4),nrow=4)
  data[,,i] = qr.Q(qr(tgt))[,1:2]
}

gdata = riemfactory(data, name="grassmann")

## Compute Pairwise Distances using pdist and pdist2
A = rbase.pdist(gdata)
B = rbase.pdist2(gdata,gdata)

## Visual Comparison in Two Cases
opar = par(no.readonly=TRUE)
par(mfrow=c(1,2), pty="s")
image(A, col=gray((0:100)/100), main="pdist")
image(B, col=gray((0:100)/100), main="pdist2")
par(opar)
```

`rbase.robust` *Robust Fréchet Mean of Manifold-valued Data*

Description

Robust estimator for mean starts from dividing the data $\{x_i\}_{i=1}^n$ into k equally sized sets. For each subset, it first estimates Fréchet mean. It then follows a step to aggregate k sample means by finding a geometric median.

Usage

```
rbase.robust(input, k = 5, maxiter = 496, eps = 1e-06, parallel = FALSE)
```

Arguments

`input`	a S3 object of `riemdata` class. See `riemfactory` for more details.
`k`	number of subsets for which the data be divided into.
`maxiter`	maximum number of iterations for gradient descent algorithm and Weiszfeld algorithm.
`eps`	stopping criterion for the norm of gradient.
`parallel`	a flag for enabling parallel computation.

Value

a named list containing

x an estimate geometric median.

iteration number of iterations until convergence.

Author(s)

Kisung You

References

Lerasle M, Oliveira R~I (2011). "Robust empirical mean Estimators." *ArXiv e-prints*. 1112.3914.

Minsker S (2013). "Geometric median and robust estimation in Banach spaces." *ArXiv e-prints*. 1308.1334.

Feng J, Xu H, Mannor S (2014). "Distributed Robust Learning." *ArXiv e-prints*. 1409.5937.

See Also

`rbase.mean`, `rbase.median`

Examples

```
### Generate 100 data points on Sphere S^2 near (0,0,1).
ndata = 100
theta = seq(from=-0.99,to=0.99,length.out=ndata)*pi
tmpx  = cos(theta) + rnorm(ndata,sd=0.1)
tmpy  = sin(theta) + rnorm(ndata,sd=0.1)

### Wrap it as 'riemdata' class
data  = list()
for (i in 1:ndata){
  tgt = c(tmpx[i],tmpy[i],1)
  data[[i]] = tgt/sqrt(sum(tgt^2)) # project onto Sphere
}
data = riemfactory(data, name="sphere")

### Compute Robust Fréchet Mean
out1 = rbase.robust(data)
out2 = rbase.robust(data,parallel=TRUE) # test parallel implementation
```

`riemfactory` *Prepare a S3 Class Object 'riemdata'*

Description

Most of the functions for `RiemBase` package require data to be wrapped as a `riemdata` class. Since manifolds of interests endow data points with specific constraints, the function `riemfactory` first checks the requirements to characterize the manifold and then wraps the data into `riemdata` class, which is simply a list of manifold-valued data and the name of manifold. Manifold name input is, fortunately, *case-insensitive*.

Usage

```
riemfactory(
  data,
  name = c("euclidean", "grassmann", "spd", "sphere", "stiefel")
)
```

Arguments

`data` data to be wrapped as `riemdata` class. Following input formats are considered,

- **2D array** an $(m \times p)$ matrix where data are stacked in columns over 2nd dimension. Appropriate for vector-valued `Euclidean` or `Sphere` manifold case.
- **3D array** an $(m \times n \times p)$ matrix where data are stacked in slices over 3rd dimension.

list unnamed list where each element of the list is a single data point. Sizes of all elements must match.

`name` the name of Riemmanian manifold for data to which data belong.

Value

a named `riemdata` S3 object containing

data a list of manifold-valued data points.

size size of each data matrix.

name name of the manifold of interests.

Examples

```
# Test with Sphere S^2 in R^3 example
## Prepare a matrix and list of 20 samples on S^2
sp.mat  = array(0,c(3,20)) # each vector will be recorded as a column
sp.list = list()
for (i in 1:20){
  tgt = rnorm(3)               # sample random numbers
  tgt = tgt/sqrt(sum(tgt*tgt)) # normalize

  sp.mat[,i]   = tgt   # record it as column vector
  sp.list[[i]] = tgt   # record it as an element in a list
}

## wrap it using 'riemfactory'
rsp1 = riemfactory(sp.mat, name="Sphere")
rsp2 = riemfactory(sp.list, name="spHeRe")
```

Index