Package ‘RiemBase’

March 21, 2020

Type Package
Title Functions and C++ Header Files for Computation on Manifolds
Version 0.2.4
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
LazyData true
Depends R (>= 3.0.0)
License GPL-3
Imports Rcpp, utils, Rdpack, parallel, stats, pracma
LinkingTo Rcpp, RcppArmadillo
RoxygenNote 7.0.2
RdMacros Rdpack
URL http://github.com/kyoustat/RiemBase
BugReports http://github.com/kyoustat/RiemBase/issues
NeedsCompilation yes
Author Kisung You [aut, cre] <https://orcid.org/0000-0002-8584-459X>}
Maintainer Kisung You <kyoustat@gmail.com>
Repository CRAN
Date/Publication 2020-03-21 19:00:27 UTC

R topics documented:

<table>
<thead>
<tr>
<th>RiemBase-package</th>
<th>rbase.curvedist</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>
**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:978-0-691-13298-3> on computational aspects of optimization on matrix manifolds.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rbase.mean</td>
<td></td>
</tr>
<tr>
<td>rbase.median</td>
<td></td>
</tr>
<tr>
<td>rbase.pdist</td>
<td></td>
</tr>
<tr>
<td>rbase.pdist2</td>
<td></td>
</tr>
<tr>
<td>rbase.robust</td>
<td></td>
</tr>
<tr>
<td>riemfactory</td>
<td></td>
</tr>
</tbody>
</table>

**Index**

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RiemBase-package</td>
<td>Functions and C++ Header Files for Computation on Manifolds</td>
</tr>
</tbody>
</table>

**rbase.curvedist**

**Description**

Suppose we have two curves \( f, g : I \subset \mathbb{R} \rightarrow \mathcal{M} \) 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**

```r
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 an equidistant sequence from 1 to \( N \).
- `type` type of Riemannian distance ("intrinsic" or "extrinsic").

**Value**

computed distance.
Examples

```r
## 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**

```r
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


Examples

```r
### 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
```
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


Examples

```r
### 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 \( x, y \in M \). Some manifolds have closed-form expression, while others need numerical approximation.

**Usage**

```r
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

```r
### 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 $Y = \{y_j\}_{j=1}^n$ 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

```r
### 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

```r
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
  - \( \mathbf{x} \) an estimate geometric median.
- iteration number of iterations until convergence.

Author(s)

Kisung You

References


See Also

- `rbase.mean`, `rbase.median`

Examples

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

```r
# 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
  sp.mat[,i] = tgt
  sp.list[[i]] = tgt
}
```
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'
rspl = riemfactory(sp.mat, name="Sphere")
rsp2 = riemfactory(sp.list, name="spHeRe")
Index

rbase.curvedist, 2
rbase.mean, 3, 9
rbase.median, 5, 9
rbase.pdist, 6, 7
rbase.pdist2, 7
rbase.robust, 8
RiemBase-package, 2
riemfactory, 3, 5, 6, 8, 10