Package 'Riemann'

September 22, 2020

Type Package

Title Learning with Data on Riemannian Manifolds

Version 0.1.0

Description We provide a variety of algorithms for manifold-valued data, including Fréchet summaries, hypothesis testing, clustering, visualization, and other learning tasks. See Bhattacharya and Bhattacharya (2012) <doi:10.1017/CBO9781139094764> for general exposition to statistics on manifolds.

License MIT + file LICENSE

Imports Rcpp (>= 1.0.5), Rdpack, RiemBase, maotai, stats, utils

Encoding UTF-8

URL http://kyoustat.com/Riemann/

BugReports https://github.com/kyoustat/Riemann/issues

LinkingTo Rcpp, RcppArmadillo

RoxygenNote 7.1.1

RdMacros Rdpack

NeedsCompilation yes

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

Maintainer Kisung You <kyoustat@gmail.com>

Repository CRAN

Date/Publication 2020-09-22 09:40:06 UTC

R topics documented:

`grassmann.runif` *Generate Uniform Samples on Grassmann Manifold*

Description

It generates n random samples from Grassmann manifold $Gr(k, p)$.

Usage

```
grassmann.runif(n, k, p, type = c("list", "array", "riemdata"))
```

Arguments

`n`	number of samples to be generated.
`k`	dimension of the subspace.
`p`	original dimension (of the ambient space).
`type`	return type;

`"list"` a length-n list of $(p \times k)$ basis of k-subspaces.
`"array"` a $(p \times k \times n)$ 3D array whose slices are k-subspace basis.
`"riemdata"` a S3 object. See `wrap.grassmann` for more details.

Value

an object from one of the above by `type` option.

References

Chikuse Y (2003). *Statistics on Special Manifolds*, volume 174 of *Lecture Notes in Statistics*. Springer New York, New York, NY. ISBN 978-0-387-00160-9 978-0-387-21540-2, doi: 10.1007/9780387215402.

See Also

`stiefel.runif`, `wrap.grassmann`

Examples

```
#-----------------------------------------------------------------
#                 Draw Samples on Grassmann Manifold
#-----------------------------------------------------------------
#  Multiple Return Types with 3 Observations of 5-dim subspaces in R^10
dat.list = grassmann.runif(n=3, k=5, p=10, type="list")
dat.arr3 = grassmann.runif(n=3, k=5, p=10, type="array")
dat.riem = grassmann.runif(n=3, k=5, p=10, type="riemdata")
```

---

grassmann.utest *Test of Uniformity on Grassmann Manifold*

Description

Given the data on Grassmann manifold $Gr(k, p)$, it tests whether the data is distributed uniformly.

Usage

```
grassmann.utest(grobj, method = c("Bing", "BingM"))
```

Arguments

`grobj`	a S3 `"riemdata"` class of Grassmann-valued data.
`method`	(case-insensitive) name of the test method containing
	`"Bing"` Bingham statistic.
	`"BingM"` modified Bingham statistic with better order of error.

Value

a (list) object of S3 class htest containing:

statistic a test statistic.

p.value p-value under H_0.

alternative alternative hypothesis.

method name of the test.

data.name name(s) of provided sample data.

References

Chikuse Y (2003). *Statistics on Special Manifolds*, volume 174 of *Lecture Notes in Statistics*. Springer New York, New York, NY. ISBN 978-0-387-00160-9 978-0-387-21540-2, doi: 10.1007/9780387215402.

Mardia KV, Jupp PE (eds.) (1999). *Directional Statistics*, Wiley Series in Probability and Statistics. John Wiley \& Sons, Inc., Hoboken, NJ, USA. ISBN 978-0-470-31697-9 978-0-471-95333-3, doi: 10.1002/9780470316979.

See Also

wrap.grassmann

Examples

```
#-------------------------------------------------------------------
#   Compare Bingham's original and modified versions of the test
#
# Test 1. sample uniformly from Gr(2,4)
# Test 2. use perturbed principal components from 'iris' data in R^4
#         which is concentrated around a point to reject H0.
#-------------------------------------------------------------------
## Data Generation
#  1. uniform data
myobj1 = grassmann.runif(n=100, k=2, p=4)

#  2. perturbed principal components
data(iris)
irdat = list()
for (n in 1:100){
   tmpdata    = iris[1:50,1:4] + matrix(rnorm(50*4,sd=0.5),ncol=4)
   irdat[[n]] = eigen(cov(tmpdata))$vectors[,1:2]
}
myobj2 = wrap.grassmann(irdat)

## Test 1 : uniform data
grassmann.utest(myobj1, method="Bing")
grassmann.utest(myobj1, method="BingM")

## Tests : iris data
grassmann.utest(myobj2, method="bINg")   # method names are
```

```
grassmann.utest(myobj2, method="BiNgM")  # CASE - INSENSITIVE !
```

`riem.clrq` *Competitive Learning Riemannian Quantization*

Description

Given N observations $X_1, X_2, \ldots, X_N \in \mathcal{M}$, perform clustering via Competitive Learning Riemannian Quantization (CLRQ). Originally, the algorithm is designed for finding voronoi cells that are used in domain quantization. Given the discrete measure of data, centers of the cells play a role of cluster centers and data are labeled accordingly based on the distance to voronoi centers. For an iterative update of centers, gradient descent algorithm adapted for the Riemannian manifold setting is used with the gain factor sequence

$$\gamma_t = \frac{a}{1 + b\sqrt{t}}$$

where two parameters a, b are represented by `par.a` and `par.b`. For initialization, we provide k-means++ and random seeding options as in k-means.

Usage

```
riem.clrq(riemobj, k = 2, init = c("plus", "random"), gain.a = 1, gain.b = 1)
```

Arguments

`riemobj`	a S3 `"riemdata"` class for N manifold-valued data.
`k`	the number of clusters.
`init`	(case-insensitive) name of an initialization scheme. (default: `"plus"`.)
`gain.a`	parameter a for gain factor sequence.
`gain.b`	parameter b for gain factor sequence.

Value

a named list containing

centers a 3d array where each slice along 3rd dimension is a matrix representation of class centers.

cluster a length-N vector of class labels (from $1:k$).

References

Le Brigant A, Puechmorel S (2019). "Quantization and clustering on Riemannian manifolds with an application to air traffic analysis." *Journal of Multivariate Analysis*, **173**, 685–703. ISSN 0047259X, doi: 10.1016/j.jmva.2019.05.008.

Bonnabel S (2013). "Stochastic Gradient Descent on Riemannian Manifolds." *IEEE Transactions on Automatic Control*, **58**(9), 2217–2229. ISSN 0018-9286, 1558-2523, doi: 10.1109/TAC.2013.2254619.

See Also

riem.kmeans

Examples

```
#-------------------------------------------------------------------
#          Example on Sphere : a dataset with three types
#
# class 1 : 10 perturbed data points near (1,0,0) on S^2 in R^3
# class 2 : 10 perturbed data points near (0,1,0) on S^2 in R^3
# class 3 : 10 perturbed data points near (0,0,1) on S^2 in R^3
#-------------------------------------------------------------------
## GENERATE DATA
mydata = list()
for (i in 1:10){
  tgt = c(1, stats::rnorm(2, sd=0.1))
  mydata[[i]] = tgt/sqrt(sum(tgt^2))
}
for (i in 11:20){
  tgt = c(rnorm(1,sd=0.1),1,rnorm(1,sd=0.1))
  mydata[[i]] = tgt/sqrt(sum(tgt^2))
}
for (i in 21:30){
  tgt = c(stats::rnorm(2, sd=0.1), 1)
  mydata[[i]] = tgt/sqrt(sum(tgt^2))
}
myriem = wrap.sphere(mydata)
mylabs = rep(c(1,2,3), each=10)

## CLRQ WITH K=2,3,4
clust2 = riem.clrq(myriem, k=2)
clust3 = riem.clrq(myriem, k=3)
clust4 = riem.clrq(myriem, k=4)

## MDS FOR VISUALIZATION
mds2d = riem.mds(myriem, ndim=2)$embed

## VISUALIZE
opar <- par(no.readonly=TRUE)
par(mfrow=c(2,2), pty="s")
plot(mds2d, pch=19, main="true label", col=mylabs)
plot(mds2d, pch=19, main="K=2", col=clust2$cluster)
plot(mds2d, pch=19, main="K=3", col=clust3$cluster)
plot(mds2d, pch=19, main="K=4", col=clust4$cluster)
par(opar)
```

---

riem.fanova *Fréchet Analysis of Variance*

Description

Given sets of manifold-valued data $X^{(1)}_{1:n_1}, X^{(2)}_{1:n_2}, \ldots, X^{(m)}_{1:n_m}$, performs analysis of variance to test equality of distributions. This means, small p-value implies that at least one of the equalities does not hold.

Usage

```
riem.fanova(..., maxiter = 50, eps = 1e-05)

riem.fanovaP(..., maxiter = 50, eps = 1e-05, nperm = 99)
```

Arguments

`...`	S3 objects of `riemdata` class for manifold-valued data.
`maxiter`	maximum number of iterations to be run.
`eps`	tolerance level for stopping criterion.
`nperm`	the number of permutations for resampling-based test.

Value

a (list) object of `S3` class `htest` containing:

statistic a test statistic.

p.value p-value under H_0.

alternative alternative hypothesis.

method name of the test.

data.name name(s) of provided sample data.

References

Dubey P, Müller H (2019). "Fréchet analysis of variance for random objects." *Biometrika*, **106**(4), 803–821. ISSN 0006-3444, 1464-3510, doi: 10.1093/biomet/asz052.

Examples

```
#-----------------------------------------------------------------
#            Example on Sphere : Uniform Samples
#
#  Each of 4 classes consists of 20 uniform samples from uniform
#  density on 2-dimensional sphere S^2 in R^3.
#-----------------------------------------------------------------
## PREPARE DATA OF 4 CLASSES
ndata  = 20
class1 = list()
class2 = list()
class3 = list()
class4 = list()
for (i in 1:ndata){
```

```
  tmp = matrix(rnorm(4*3), ncol=3)
  tmp = tmp/sqrt(rowSums(tmp^2))

  class1[[i]] = tmp[1,]
  class2[[i]] = tmp[2,]
  class3[[i]] = tmp[3,]
  class4[[i]] = tmp[4,]
}
obj1 = wrap.sphere(class1)
obj2 = wrap.sphere(class2)
obj3 = wrap.sphere(class3)
obj4 = wrap.sphere(class4)

## RUN THE ASYMPTOTIC TEST
riem.fanova(obj1, obj2, obj3, obj4)


## RUN THE PERMUTATION TEST WITH MANY PERMUTATIONS
riem.fanovaP(obj1, obj2, obj3, obj4, nperm=9999)
```

`riem.hclust` *Hierarchical Agglomerative Clustering*

Description

Given N observations $X_1, X_2, \ldots, X_M \in \mathcal{M}$, perform hierarchical agglomerative clustering with **fastcluster** package's implementation.

Usage

```
riem.hclust(
  riemobj,
  geometry = c("intrinsic", "extrinsic"),
  method = c("single", "complete", "average", "mcquitty", "ward.D", "ward.D2",
    "centroid", "median"),
  members = NULL
)
```

Arguments

`riemobj`	a S3 `"riemdata"` class for N manifold-valued data.
`geometry`	(case-insensitive) name of geometry; either geodesic (`"intrinsic"`) or embedded (`"extrinsic"`) geometry.
`method`	agglomeration method to be used. This must be one of `"single"`, `"complete"`, `"average"`, `"mcquitty"`, `"ward.D"`, `"ward.D2"`, `"centroid"` or `"median"`.
`members`	`NULL` or a vector whose length equals the number of observations. See `hclust` for details.

Value

an object of class `hclust`. See `hclust` for details.

See Also

`hclust`

Examples

```
#-----------------------------------------------------------------
#          Example on Sphere : a dataset with three types
#
# class 1 : 10 perturbed data points near (1,0,0) on S^2 in R^3
# class 2 : 10 perturbed data points near (0,1,0) on S^2 in R^3
# class 3 : 10 perturbed data points near (0,0,1) on S^2 in R^3
#-----------------------------------------------------------------
## GENERATE DATA
mydata = list()
for (i in 1:10){
  tgt = c(1, stats::rnorm(2, sd=0.1))
  mydata[[i]] = tgt/sqrt(sum(tgt^2))
}
for (i in 11:20){
  tgt = c(rnorm(1,sd=0.1),1,rnorm(1,sd=0.1))
  mydata[[i]] = tgt/sqrt(sum(tgt^2))
}
for (i in 21:30){
  tgt = c(stats::rnorm(2, sd=0.1), 1)
  mydata[[i]] = tgt/sqrt(sum(tgt^2))
}
myriem = wrap.sphere(mydata)

## COMPUTE SINGLE AND COMPLETE LINKAGE
hc.sing <- riem.hclust(myriem, method="single")
hc.comp <- riem.hclust(myriem, method="complete")

## VISUALIZE
opar <- par(no.readonly=TRUE)
par(mfrow=c(1,2))
plot(hc.sing, main="single linkage")
plot(hc.comp, main="complete linkage")
par(opar)
```

`riem.interp` *Geodesic Interpolation*

Description

Given 2 observations $X_1, X_2 \in \mathcal{M}$, find the interpolated point of a geodesic $\gamma(t)$ for $t \in (0,1)$ which assumes two endpoints $\gamma(0) = X_1$ and $\gamma(1) = X_2$.

Usage

```
riem.interp(riemobj, t = 0.5, geometry = c("intrinsic", "extrinsic"))
```

Arguments

riemobj	a S3 "riemdata" class for 2 manifold-valued data where the first object is the starting point.
t	a scalar in $(0, 1)$ for which the interpolation is taken.
geometry	(case-insensitive) name of geometry; either geodesic ("intrinsic") or embedded ("extrinsic") geometry.

Value

an interpolated object in matrix representation on $\mathcal{M}$.

Examples

```
#-----------------------------------------------------------------
#       Geodesic Interpolation between (1,0) and (0,1) in S^1
#-----------------------------------------------------------------
## PREPARE DATA
sp.start = c(1,0)
sp.end   = c(0,1)
sp.data  = wrap.sphere(rbind(sp.start, sp.end))

## FIND THE INTERPOLATED POINT AT "t=0.25"
mid.int = as.vector(riem.interp(sp.data, t=0.25, geometry="intrinsic"))
mid.ext = as.vector(riem.interp(sp.data, t=0.25, geometry="extrinsic"))

## VISUALIZE
#  Prepare Lines and Points
thetas  = seq(from=0, to=pi/2, length.out=100)
quarter = cbind(cos(thetas), sin(thetas))
pic.pts = rbind(sp.start, mid.int, mid.ext, sp.end)
pic.col = c("black","red","green","black")

# Draw
opar <- par(no.readonly=TRUE)
par(pty="s")
plot(quarter, main="two interpolated points at t=0.25",
     xlab="x", ylab="y", type="l")
points(pic.pts, col=pic.col, pch=19)
text(mid.int[1]-0.1, mid.int[2], "intrinsic", col="red")
text(mid.ext[1]-0.1, mid.ext[2], "extrinsic", col="green")
par(opar)
```

`riem.interps` *Geodesic Interpolation of Multiple Points*

Description

Given 2 observations $X_1, X_2 \in \mathcal{M}$, find the interpolated points of a geodesic $\gamma(t)$ for $t \in (0,1)$ which assumes two endpoints $\gamma(0) = X_1$ and $\gamma(1) = X_2$.

Usage

```
riem.interps(
  riemobj,
  vect = c(0.25, 0.5, 0.75),
  geometry = c("intrinsic", "extrinsic")
)
```

Arguments

`riemobj`	a S3 `"riemdata"` class for 2 manifold-valued data where the first object is the starting point.
`vect`	a length-T vector in $(0,1)$ for which the interpolations are taken.
`geometry`	(case-insensitive) name of geometry; either geodesic (`"intrinsic"`) or embedded (`"extrinsic"`) geometry.

Value

a 3d array where T slices along 3rd dimension are interpolated objects in matrix representation.

Examples

```
#-----------------------------------------------------------------
#       Geodesic Interpolation between (1,0) and (0,1) in S^1
#-----------------------------------------------------------------
## PREPARE DATA
sp.start = c(1,0)
sp.end   = c(0,1)
sp.data  = wrap.sphere(rbind(sp.start, sp.end))

## FIND THE INTERPOLATED POINT AT FOR t=0.1, 0.2, ..., 0.9.
myvect  = seq(from=0.1, to=0.9, by=0.1)
geo.int = riem.interps(sp.data, vect=myvect, geometry="intrinsic")
geo.ext = riem.interps(sp.data, vect=myvect, geometry="extrinsic")

geo.int = matrix(geo.int, byrow=TRUE, ncol=2) # re-arrange for plotting
geo.ext = matrix(geo.ext, byrow=TRUE, ncol=2)

## VISUALIZE
#  Prepare Lines and Points
```

```
thetas  = seq(from=0, to=pi/2, length.out=100)
quarter = cbind(cos(thetas), sin(thetas))

pts.int = rbind(sp.start, geo.int, sp.end)
pts.ext = rbind(sp.start, geo.ext, sp.end)
col.int = c("black", rep("red",9),  "black")
col.ext = c("black", rep("blue",9), "black")

# Draw
opar <- par(no.readonly=TRUE)
par(mfrow=c(1,2), pty="s")
plot(quarter, main="intrinsic interpolation", # intrinsic geodesic
     xlab="x", ylab="y", type="l")
points(pts.int, col=col.int, pch=19)
for (i in 1:9){
  text(geo.int[i,1]*0.9, geo.int[i,2]*0.9,
       paste0(round(i/10,2)), col="red")
}
plot(quarter, main="extrinsic interpolation", # intrinsic geodesic
     xlab="x", ylab="y", type="l")
points(pts.ext, col=col.ext, pch=19)
for (i in 1:9){
  text(geo.ext[i,1]*0.9, geo.ext[i,2]*0.9,
       paste0(round(i/10,2)), col="blue")
}
par(opar)
```

`riem.kmeans` *K-Means Clustering*

Description

Given N observations $X_1, X_2, \ldots, X_N \in \mathcal{M}$, perform k-means clustering by minimizing within-cluster sum of squares (WCSS). Since the problem is NP-hard and sensitive to the initialization, we provide an option with multiple starts and return the best result with respect to WCSS.

Usage

```
riem.kmeans(
  riemobj,
  k = 2,
  geometry = c("intrinsic", "extrinsic"),
  maxiter = 10,
  nstart = 5,
  algorithm = c("MacQueen", "Lloyd"),
  init = c("plus", "random")
)
```

Arguments

`riemobj`	a S3 `"riemdata"` class for N manifold-valued data.
`k`	the number of clusters.
`geometry`	(case-insensitive) name of geometry; either geodesic (`"intrinsic"`) or embedded (`"extrinsic"`) geometry.
`maxiter`	the maximum number of iterations allowed.
`nstart`	the number of random starts.
`algorithm`	(case-insensitive) name of an algorithm to be run. (default: `"MacQueen"`)
`init`	(case-insensitive) name of an initialization scheme. (default: `"plus"`)

Value

a named list containing

means a 3d array where each slice along 3rd dimension is a matrix representation of class mean.

cluster a length-N vector of class labels (from $1:k$).

score within-cluster sum of squares (WCSS).

References

Lloyd S (1982). "Least squares quantization in PCM." *IEEE Transactions on Information Theory*, **28**(2), 129–137. ISSN 0018-9448, doi: 10.1109/TIT.1982.1056489.

MacQueen J (1967). "Some methods for classification and analysis of multivariate observations." In *Proceedings of the fifth berkeley symposium on mathematical statistics and probability, volume 1: Statistics*, 281–297.

See Also

`riem.kmeanspp`

Examples

```
#-----------------------------------------------------------------
#          Example on Sphere : a dataset with three types
#
# class 1 : 10 perturbed data points near (1,0,0) on S^2 in R^3
# class 2 : 10 perturbed data points near (0,1,0) on S^2 in R^3
# class 3 : 10 perturbed data points near (0,0,1) on S^2 in R^3
#-----------------------------------------------------------------
## GENERATE DATA
mydata = list()
for (i in 1:10){
  tgt = c(1, stats::rnorm(2, sd=0.1))
  mydata[[i]] = tgt/sqrt(sum(tgt^2))
}
for (i in 11:20){
  tgt = c(rnorm(1,sd=0.1),1,rnorm(1,sd=0.1))
  mydata[[i]] = tgt/sqrt(sum(tgt^2))
```

```
}
for (i in 21:30){
  tgt = c(stats::rnorm(2, sd=0.1), 1)
  mydata[[i]] = tgt/sqrt(sum(tgt^2))
}
myriem = wrap.sphere(mydata)
mylabs = rep(c(1,2,3), each=10)

## K-MEDOIDS WITH K=2,3,4
clust2 = riem.kmeans(myriem, k=2)
clust3 = riem.kmeans(myriem, k=3)
clust4 = riem.kmeans(myriem, k=4)

## MDS FOR VISUALIZATION
mds2d = riem.mds(myriem, ndim=2)$embed

## VISUALIZE
opar <- par(no.readonly=TRUE)
par(mfrow=c(2,2), pty="s")
plot(mds2d, pch=19, main="true label", col=mylabs)
plot(mds2d, pch=19, main="K=2", col=clust2$cluster)
plot(mds2d, pch=19, main="K=3", col=clust3$cluster)
plot(mds2d, pch=19, main="K=4", col=clust4$cluster)
par(opar)
```

`riem.kmeanspp` *K-Means++ Clustering*

Description

Given N observations $X_1, X_2, \ldots, X_N \in \mathcal{M}$, perform k-means++ clustering algorithm using pairwise distances. The algorithm was originally designed as an efficient initialization method for k-means algorithm.

Usage

```
riem.kmeanspp(riemobj, k = 2, geometry = c("intrinsic", "extrinsic"))
```

Arguments

`riemobj`	a S3 `"riemdata"` class for N manifold-valued data.
`k`	the number of clusters.
`geometry`	(case-insensitive) name of geometry; either geodesic (`"intrinsic"`) or embedded (`"extrinsic"`) geometry.

Value

a named list containing

centers a length-k vector of sampled centers' indices.

cluster a length-N vector of class labels (from $1:k$).

References

Arthur D, Vassilvitskii S (2007). "K-Means++: The advantages of careful seeding." In *Proceedings of the eighteenth annual ACM-SIAM symposium on discrete algorithms*, SODA '07, 1027–1035. ISBN 978-0-89871-624-5, Number of pages: 9 Place: New Orleans, Louisiana.

Examples

```
#-----------------------------------------------------------------
#          Example on Sphere : a dataset with three types
#
# class 1 : 10 perturbed data points near (1,0,0) on S^2 in R^3
# class 2 : 10 perturbed data points near (0,1,0) on S^2 in R^3
# class 3 : 10 perturbed data points near (0,0,1) on S^2 in R^3
#-----------------------------------------------------------------
## GENERATE DATA
mydata = list()
for (i in 1:10){
  tgt = c(1, stats::rnorm(2, sd=0.1))
  mydata[[i]] = tgt/sqrt(sum(tgt^2))
}
for (i in 11:20){
  tgt = c(rnorm(1,sd=0.1),1,rnorm(1,sd=0.1))
  mydata[[i]] = tgt/sqrt(sum(tgt^2))
}
for (i in 21:30){
  tgt = c(stats::rnorm(2, sd=0.1), 1)
  mydata[[i]] = tgt/sqrt(sum(tgt^2))
}
myriem = wrap.sphere(mydata)
mylabs = rep(c(1,2,3), each=10)

## K-MEANS++ WITH K=2,3,4
clust2 = riem.kmeanspp(myriem, k=2)
clust3 = riem.kmeanspp(myriem, k=3)
clust4 = riem.kmeanspp(myriem, k=4)

## MDS FOR VISUALIZATION
mds2d = riem.mds(myriem, ndim=2)$embed

## VISUALIZE
opar <- par(no.readonly=TRUE)
par(mfrow=c(2,2), pty="s")
plot(mds2d, pch=19, main="true label", col=mylabs)
plot(mds2d, pch=19, main="K=2", col=clust2$cluster)
plot(mds2d, pch=19, main="K=3", col=clust3$cluster)
```

```
plot(mds2d, pch=19, main="K=4", col=clust4$cluster)
par(opar)
```

`riem.kmedoids` *K-Medoids Clustering*

Description

Given N observations $X_1, X_2, \ldots, X_N \in \mathcal{M}$, perform k-medoids clustering using pairwise distances.

Usage

```
riem.kmedoids(riemobj, k = 2, geometry = c("intrinsic", "extrinsic"))
```

Arguments

`riemobj`	a S3 `"riemdata"` class for N manifold-valued data.
`k`	the number of clusters.
`geometry`	(case-insensitive) name of geometry; either geodesic (`"intrinsic"`) or embedded (`"extrinsic"`) geometry.

Value

a named list containing

medoids a length-k vector of medoids' indices.

cluster a length-N vector of class labels (from $1:k$).

See Also

`pam`

Examples

```
#-----------------------------------------------------------------
#          Example on Sphere : a dataset with three types
#
# class 1 : 10 perturbed data points near (1,0,0) on S^2 in R^3
# class 2 : 10 perturbed data points near (0,1,0) on S^2 in R^3
# class 3 : 10 perturbed data points near (0,0,1) on S^2 in R^3
#-----------------------------------------------------------------
## GENERATE DATA
mydata = list()
for (i in 1:10){
  tgt = c(1, stats::rnorm(2, sd=0.1))
  mydata[[i]] = tgt/sqrt(sum(tgt^2))
```

```
}
for (i in 11:20){
  tgt = c(rnorm(1,sd=0.1),1,rnorm(1,sd=0.1))
  mydata[[i]] = tgt/sqrt(sum(tgt^2))
}
for (i in 21:30){
  tgt = c(stats::rnorm(2, sd=0.1), 1)
  mydata[[i]] = tgt/sqrt(sum(tgt^2))
}
myriem = wrap.sphere(mydata)
mylabs = rep(c(1,2,3), each=10)

## K-MEDOIDS WITH K=2,3,4
clust2 = riem.kmedoids(myriem, k=2)
clust3 = riem.kmedoids(myriem, k=3)
clust4 = riem.kmedoids(myriem, k=4)

## MDS FOR VISUALIZATION
mds2d = riem.mds(myriem, ndim=2)$embed

## VISUALIZE
opar <- par(no.readonly=TRUE)
par(mfrow=c(2,2), pty="s")
plot(mds2d, pch=19, main="true label", col=mylabs)
plot(mds2d, pch=19, main="K=2", col=clust2$cluster)
plot(mds2d, pch=19, main="K=3", col=clust3$cluster)
plot(mds2d, pch=19, main="K=4", col=clust4$cluster)
par(opar)
```

`riem.knn` *Find K-Nearest Neighbors*

Description

Given N observations $X_1, X_2, \ldots, X_N \in \mathcal{M}$, `riem.knn` constructs k-nearest neighbors. This is a wrapper for a main function in **nabor** package. Note that the original function contains index for each data point itself, but our function does not consider self-neighborhood scenario.

Usage

```
riem.knn(riemobj, k = 2, geometry = c("intrinsic", "extrinsic"), ...)
```

Arguments

`riemobj`	a S3 `"riemdata"` class for N manifold-valued data.
`k`	the number of neighbors to find.
`geometry`	(case-insensitive) name of geometry; either geodesic (`"intrinsic"`) or embedded (`"extrinsic"`) geometry.
`...`	extra parameters to be passed onto knn function.

Value

a named list containing

nn.idx an $(N \times k)$ neighborhood index matrix.

nn.dists an $(N \times k)$ distances from a point to its neighbors.

See Also

knn

Examples

```
#-----------------------------------------------------------------
#          Example on Sphere : a dataset with three types
#
# * 10 perturbed data points near (1,0,0) on S^2 in R^3
# * 10 perturbed data points near (0,1,0) on S^2 in R^3
# * 10 perturbed data points near (0,0,1) on S^2 in R^3
#-----------------------------------------------------------------
## GENERATE DATA
mydata = list()
for (i in 1:10){
  tgt = c(1, stats::rnorm(2, sd=0.1))
  mydata[[i]] = tgt/sqrt(sum(tgt^2))
}
for (i in 11:20){
  tgt = c(rnorm(1,sd=0.1),1,rnorm(1,sd=0.1))
  mydata[[i]] = tgt/sqrt(sum(tgt^2))
}
for (i in 21:30){
  tgt = c(stats::rnorm(2, sd=0.1), 1)
  mydata[[i]] = tgt/sqrt(sum(tgt^2))
}
myriem = wrap.sphere(mydata)
mylabs = rep(c(2,3,4), each=10)

## K-NN CONSTRUCTION WITH K=5 & K=10
knn1 = riem.knn(myriem, k=5)
knn2 = riem.knn(myriem, k=10)

## MDS FOR VISUALIZATION
embed2 = riem.mds(myriem, ndim=2)$embed

## VISUALIZE
opar <- par(no.readonly=TRUE)
par(mfrow=c(1,2), pty="s")
plot(embed2, pch=19, main="knn with k=4", col=mylabs)
for (i in 1:30){
  for (j in 1:5){
    lines(embed2[c(i,knn1$nn.idx[i,j]),])
  }
}
```

```
plot(embed2, pch=19, main="knn with k=8", col=mylabs)
for (i in 1:30){
  for (j in 1:10){
    lines(embed2[c(i,knn2$nn.idx[i,j]),])
  }
}
par(opar)
```

`riem.mds` *Multidimensional Scaling*

Description

Given N observations $X_1, X_2, \ldots, X_N \in \mathcal{M}$, apply multidimensional scaling to get low-dimensional embedding in Euclidean space. Usually, `ndim=2,3` are chosen for visualization.

Usage

```
riem.mds(riemobj, ndim = 2, geometry = c("intrinsic", "extrinsic"))
```

Arguments

`riemobj`	a S3 `"riemdata"` class for N manifold-valued data.
`ndim`	an integer-valued target dimension.
`geometry`	(case-insensitive) name of geometry; either geodesic (`"intrinsic"`) or embedded (`"extrinsic"`) geometry.

Value

a named list containing

embed an $(N \times ndim)$ matrix whose rows are embedded observations.

stress discrepancy between embedded and original distances as a measure of error.

References

Torgerson WS (1952). "Multidimensional scaling: I. Theory and method." *Psychometrika*, **17**(4), 401–419. ISSN 0033-3123, 1860-0980, doi: 10.1007/BF02288916.

Examples

```
#-----------------------------------------------------------------
#          Example on Sphere : a dataset with three types
#
# 10 perturbed data points near (1,0,0) on S^2 in R^3
# 10 perturbed data points near (0,1,0) on S^2 in R^3
# 10 perturbed data points near (0,0,1) on S^2 in R^3
```

```
#-----------------------------------------------------------------
## GENERATE DATA
mydata = list()
for (i in 1:10){
  tgt = c(1, stats::rnorm(2, sd=0.1))
  mydata[[i]] = tgt/sqrt(sum(tgt^2))
}
for (i in 11:20){
  tgt = c(rnorm(1,sd=0.1),1,rnorm(1,sd=0.1))
  mydata[[i]] = tgt/sqrt(sum(tgt^2))
}
for (i in 21:30){
  tgt = c(stats::rnorm(2, sd=0.1), 1)
  mydata[[i]] = tgt/sqrt(sum(tgt^2))
}
myriem = wrap.sphere(mydata)
mylabs = rep(c(1,2,3), each=10)

## MDS EMBEDDING WITH TWO GEOMETRIES
embed2int = riem.mds(myriem, geometry="intrinsic")$embed
embed2ext = riem.mds(myriem, geometry="extrinsic")$embed

## VISUALIZE
opar = par(no.readonly=TRUE)
par(mfrow=c(1,2), pty="s")
plot(embed2int, main="intrinsic MDS", ylim=c(-2,2), col=mylabs, pch=19)
plot(embed2ext, main="extrinsic MDS", ylim=c(-2,2), col=mylabs, pch=19)
par(opar)
```

`riem.mean` *Fréchet Mean and Variation*

Description

Given N observations $X_1, X_2, \ldots, X_N \in \mathcal{M}$, compute Fréchet mean and variation with respect to the geometry by minimizing

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

where $\rho(x, y)$ is a distance for two points $x, y \in \mathcal{M}$. If non-uniform weights are given, normalized version of the mean is computed and if `weight=NULL`, it automatically sets equal weights for all observations.

Usage

```
riem.mean(
  riemobj,
```

```
  weight = NULL,
  geometry = c("intrinsic", "extrinsic"),
  maxiter = 50,
  eps = 1e-05
)
```

Arguments

riemobj	a S3 "riemdata" class for N manifold-valued data.
weight	weight of observations; if NULL it assumes equal weights, or a nonnegative length-N vector that sums to 1 should be given.
geometry	(case-insensitive) name of geometry; either geodesic ("intrinsic") or embedded ("extrinsic") geometry.
maxiter	maximum number of iterations to be run.
eps	tolerance level for stopping criterion.

Value

a named list containing

mean a mean matrix on $\mathcal{M}$.

variation sum of (weighted) squared distances.

Examples

```
#------------------------------------------------------------------
#        Example on Sphere : points near (0,1) on S^1 in R^2
#------------------------------------------------------------------
## GENERATE DATA
ndata = 50
mydat = array(0,c(ndata,2))
for (i in 1:ndata){
  tgt = c(stats::rnorm(1, sd=2), 1)
  mydat[i,] = tgt/sqrt(sum(tgt^2))
}
myriem = wrap.sphere(mydat)

## COMPUTE TWO MEANS
mean.int = as.vector(riem.mean(myriem, geometry="intrinsic")$mean)
mean.ext = as.vector(riem.mean(myriem, geometry="extrinsic")$mean)

## VISUALIZE
opar <- par(no.readonly=TRUE)
plot(mydat[,1], mydat[,2], pch=19, xlim=c(-1.1,1.1), ylim=c(0,1.1),
     main="BLUE-extrinsic vs RED-intrinsic")
arrows(x0=0,y0=0,x1=mean.int[1],y1=mean.int[2],col="red")
arrows(x0=0,y0=0,x1=mean.ext[1],y1=mean.ext[2],col="blue")
par(opar)
```

`riem.median` *Fréchet Median and Variation*

Description

Given N observations $X_1, X_2, \ldots, X_N \in \mathcal{M}$, compute Fréchet median and variation with respect to the geometry by minimizing

$$\min_x \sum_{n=1}^{N} w_n \rho(x, x_n), \quad x \in \mathcal{M}$$

where $\rho(x, y)$ is a distance for two points $x, y \in \mathcal{M}$. If non-uniform weights are given, normalized version of the mean is computed and if `weight=NULL`, it automatically sets equal weights for all observations.

Usage

```
riem.median(
  riemobj,
  weight = NULL,
  geometry = c("intrinsic", "extrinsic"),
  maxiter = 50,
  eps = 1e-05
)
```

Arguments

`riemobj`	a S3 `"riemdata"` class for N manifold-valued data.
`weight`	weight of observations; if `NULL` it assumes equal weights, or a nonnegative length-N vector that sums to 1 should be given.
`geometry`	(case-insensitive) name of geometry; either geodesic (`"intrinsic"`) or embedded (`"extrinsic"`) geometry.
`maxiter`	maximum number of iterations to be run.
`eps`	tolerance level for stopping criterion.

Value

a named list containing

median a median matrix on $\mathcal{M}$.

variation sum of (weighted) distances.

Examples

```
#-----------------------------------------------------------------
#        Example on Sphere : points near (0,1) on S^1 in R^2
#-----------------------------------------------------------------
## GENERATE DATA
ndata = 50
mydat = array(0,c(ndata,2))
for (i in 1:ndata){
  tgt = c(stats::rnorm(1, sd=2), 1)
  mydat[i,] = tgt/sqrt(sum(tgt^2))
}
myriem = wrap.sphere(mydat)

## COMPUTE TWO MEANS
med.int = as.vector(riem.median(myriem, geometry="intrinsic")$median)
med.ext = as.vector(riem.median(myriem, geometry="extrinsic")$median)

## VISUALIZE
opar <- par(no.readonly=TRUE)
plot(mydat[,1], mydat[,2], pch=19, xlim=c(-1.1,1.1), ylim=c(0,1.1),
     main="BLUE-extrinsic vs RED-intrinsic")
arrows(x0=0,y0=0,x1=med.int[1],y1=med.int[2],col="red")
arrows(x0=0,y0=0,x1=med.ext[1],y1=med.ext[2],col="blue")
par(opar)
```

`riem.nmshift` *Nonlinear Mean Shift*

Description

Given N observations $X_1, X_2, \ldots, X_N \in \mathcal{M}$, perform clustering of the data based on the nonlinear mean shift algorithm. Gaussian kernel is used with the bandwidth h as of

$$G(x_i, x_j) \propto \exp\left(-\frac{\rho^2(x_i, x_j)}{h^2}\right)$$

where $\rho(x, y)$ is geodesic distance between two points $x, y \in \mathcal{M}$. Numerically, some of the limiting points that collapse into the same cluster are not exact. For such purpose, we require `maxk` parameter to search the optimal number of clusters based on k-medoids clustering algorithm in conjunction with silhouette criterion.

Usage

```
riem.nmshift(riemobj, h = 1, maxk = 5, maxiter = 50, eps = 1e-05)
```

Arguments

`riemobj`	a S3 `"riemdata"` class for N manifold-valued data.
`h`	bandwidth parameter. The larger the h is, the more blurring is applied.
`maxk`	maximum number of clusters to determine the optimal number of clusters.
`maxiter`	maximum number of iterations to be run.
`eps`	tolerance level for stopping criterion.

Value

a named list containing

distance an $(N \times N)$ distance between modes corresponding to each data point.

cluster a length-N vector of class labels.

References

Subbarao R, Meer P (2009). "Nonlinear Mean Shift over Riemannian Manifolds." *International Journal of Computer Vision*, **84**(1), 1–20. ISSN 0920-5691, 1573-1405, doi: 10.1007/s11263008-01958.

Examples

```
#-----------------------------------------------------------------
#          Example on Sphere : a dataset with three types
#
# class 1 : 10 perturbed data points near (1,0,0) on S^2 in R^3
# class 2 : 10 perturbed data points near (0,1,0) on S^2 in R^3
# class 3 : 10 perturbed data points near (0,0,1) on S^2 in R^3
#-----------------------------------------------------------------
## GENERATE DATA
set.seed(496)
ndata  = 10
mydata = list()
for (i in 1:ndata){
  tgt = c(1, stats::rnorm(2, sd=0.1))
  mydata[[i]] = tgt/sqrt(sum(tgt^2))
}
for (i in (ndata+1):(2*ndata)){
  tgt = c(rnorm(1,sd=0.1),1,rnorm(1,sd=0.1))
  mydata[[i]] = tgt/sqrt(sum(tgt^2))
}
for (i in ((2*ndata)+1):(3*ndata)){
  tgt = c(stats::rnorm(2, sd=0.1), 1)
  mydata[[i]] = tgt/sqrt(sum(tgt^2))
}
myriem = wrap.sphere(mydata)
mylabs = rep(c(1,2,3), each=ndata)

## RUN NONLINEAR MEANSHIFT FOR DIFFERENT 'h' VALUES
run1 = riem.nmshift(myriem, maxk=10, h=0.1)
```

```
run2 = riem.nmshift(myriem, maxk=10, h=1)
run3 = riem.nmshift(myriem, maxk=10, h=10)

## MDS FOR VISUALIZATION
mds2d = riem.mds(myriem, ndim=2)$embed

## VISUALIZE
opar <- par(no.readonly=TRUE)
par(mfrow=c(2,3), pty="s")
plot(mds2d, pch=19, main="label : h=0.1", col=run1$cluster)
plot(mds2d, pch=19, main="label : h=1",   col=run2$cluster)
plot(mds2d, pch=19, main="label : h=10",  col=run3$cluster)
image(run1$distance[,30:1], axes=FALSE, main="distance : h=0.1")
image(run2$distance[,30:1], axes=FALSE, main="distance : h=1")
image(run3$distance[,30:1], axes=FALSE, main="distance : h=10")
par(opar)
```

`riem.pdist` *Compute Pairwise Distances for Data*

Description

Given N observations $X_1, X_2, \ldots, X_N \in \mathcal{M}$, compute pairwise distances.

Usage

```
riem.pdist(riemobj, geometry = c("intrinsic", "extrinsic"), as.dist = FALSE)
```

Arguments

`riemobj`	a S3 `"riemdata"` class for N manifold-valued data.
`geometry`	(case-insensitive) name of geometry; either geodesic (`"intrinsic"`) or embedded (`"extrinsic"`) in geometry
`as.dist`	logical; if `TRUE`, it returns `dist` object, else it returns a symmetric matrix.

Value

a S3 `dist` object or $(N \times N)$ symmetric matrix of pairwise distances according to `as.dist` parameter.

Examples

```
#-----------------------------------------------------------------
#          Example on Sphere : a dataset with two types
#
#  group1 : perturbed data points near (0,0,1) on S^2 in R^3
#  group2 : perturbed data points near (1,0,0) on S^2 in R^3
#-----------------------------------------------------------------
```

```
## GENERATE DATA
mydata = list()
sdval  = 0.1
for (i in 1:10){
  tgt = c(stats::rnorm(2, sd=sdval), 1)
  mydata[[i]] = tgt/sqrt(sum(tgt^2))
}
for (i in 11:20){
  tgt = c(1, stats::rnorm(2, sd=sdval))
  mydata[[i]] = tgt/sqrt(sum(tgt^2))
}
myriem = wrap.sphere(mydata)

## COMPARE TWO DISTANCES
dint = riem.pdist(myriem, geometry="intrinsic", as.dist=FALSE)
dext = riem.pdist(myriem, geometry="extrinsic", as.dist=FALSE)

## VISUALIZE
opar = par(no.readonly=TRUE)
par(mfrow=c(1,2), pty="s")
image(dint[,nrow(dint):1], main="intrinsic", axes=FALSE)
image(dext[,nrow(dext):1], main="extrinsic", axes=FALSE)
par(opar)
```

`riem.pdist2` *Compute Pairwise Distances for Two Sets of Data*

Description

Given M observations $X_1, X_2, \ldots, X_M \in \mathcal{M}$ and N observations $Y_1, Y_2, \ldots, Y_N \in \mathcal{M}$, compute pairwise distances between two sets' elements.

Usage

```
riem.pdist2(riemobj1, riemobj2, geometry = c("intrinsic", "extrinsic"))
```

Arguments

`riemobj1`	a S3 `"riemdata"` class for M manifold-valued data.
`riemobj2`	a S3 `"riemdata"` class for N manifold-valued data.
`geometry`	(case-insensitive) name of geometry; either geodesic (`"intrinsic"`) or embedded (`"extrinsic"`) geometry.

Value

an $(M \times N)$ matrix of distances.

Examples

```
#------------------------------------------------------------------
#          Example on Sphere : a dataset with two types
#
#  group1 : 10 perturbed data points near (0,0,1) on S^2 in R^3
#  group2 : 10 perturbed data points near (1,0,0) on S^2 in R^3
#           10 perturbed data points near (0,1,0) on S^2 in R^3
#           10 perturbed data points near (0,0,1) on S^2 in R^3
#------------------------------------------------------------------
## GENERATE DATA
mydata1 = list()
mydata2 = list()
for (i in 1:10){
  tgt = c(stats::rnorm(2, sd=0.1), 1)
  mydata1[[i]] = tgt/sqrt(sum(tgt^2))
}
for (i in 1:10){
  tgt = c(1, stats::rnorm(2, sd=0.1))
  mydata2[[i]] = tgt/sqrt(sum(tgt^2))
}
for (i in 11:20){
  tgt = c(rnorm(1,sd=0.1),1,rnorm(1,sd=0.1))
  mydata2[[i]] = tgt/sqrt(sum(tgt^2))
}
for (i in 21:30){
  tgt = c(stats::rnorm(2, sd=0.1), 1)
  mydata2[[i]] = tgt/sqrt(sum(tgt^2))
}
myriem1 = wrap.sphere(mydata1)
myriem2 = wrap.sphere(mydata2)

## COMPARE TWO DISTANCES
dint = riem.pdist2(myriem1, myriem2, geometry="intrinsic")
dext = riem.pdist2(myriem1, myriem2, geometry="extrinsic")

## VISUALIZE
opar = par(no.readonly=TRUE)
par(mfrow=c(1,2))
image(dint[nrow(dint):1,], main="intrinsic", axes=FALSE)
image(dext[nrow(dext):1,], main="extrinsic", axes=FALSE)
par(opar)
```

`riem.pga` *Principal Geodesic Analysis*

Description

Given N observations $X_1, X_2, \ldots, X_N \in \mathcal{M}$, Principal Geodesic Analysis (PGA) finds a low-dimensional embedding by decomposing 2nd-order information in tangent space at an intrinsic mean of the data.

Usage

```
riem.pga(riemobj, ndim = 2)
```

Arguments

riemobj	a S3 "riemdata" class for N manifold-valued data.
ndim	an integer-valued target dimension.

Value

a named list containing

center an intrinsic mean in a matrix representation form.

embed an $(N \times ndim)$ matrix whose rows are embedded observations.

References

Fletcher P, Lu C, Pizer S, Joshi S (2004). "Principal Geodesic Analysis for the Study of Nonlinear Statistics of Shape." *IEEE Transactions on Medical Imaging*, **23**(8), 995–1005. ISSN 0278-0062, doi: 10.1109/TMI.2004.831793.

Examples

```
#------------------------------------------------------------------
#          Example on Sphere : a dataset with three types
#
# 10 perturbed data points near (1,0,0) on S^2 in R^3
# 10 perturbed data points near (0,1,0) on S^2 in R^3
# 10 perturbed data points near (0,0,1) on S^2 in R^3
#------------------------------------------------------------------
## GENERATE DATA
mydata = list()
for (i in 1:10){
  tgt = c(1, stats::rnorm(2, sd=0.1))
  mydata[[i]] = tgt/sqrt(sum(tgt^2))
}
for (i in 11:20){
  tgt = c(rnorm(1,sd=0.1),1,rnorm(1,sd=0.1))
  mydata[[i]] = tgt/sqrt(sum(tgt^2))
}
for (i in 21:30){
  tgt = c(stats::rnorm(2, sd=0.1), 1)
  mydata[[i]] = tgt/sqrt(sum(tgt^2))
}
myriem = wrap.sphere(mydata)
mylabs = rep(c(1,2,3), each=10)

## EMBEDDING WITH MDS AND PGA
embed2mds = riem.mds(myriem, ndim=2, geometry="intrinsic")$embed
embed2pga = riem.pga(myriem, ndim=2)$embed
```

```
## VISUALIZE
opar = par(no.readonly=TRUE)
par(mfrow=c(1,2), pty="s")
plot(embed2mds, main="Multidimensional Scaling",    col=mylabs, pch=19)
plot(embed2pga, main="Principal Geodesic Analysis", col=mylabs, pch=19)
par(opar)
```

`riem.rmml` *Riemannian Manifold Metric Learning*

Description

Given N observations $X_1, X_2, \ldots, X_N \in \mathcal{M}$ and corresponding label information, `riem.rmml` computes pairwise distance of data under Riemannian Manifold Metric Learning (RMML) framework based on equivariant embedding. When the number of data points is not sufficient, an inverse of scatter matrix does not exist analytically so the small regularization parameter λ is recommended with default value of $\lambda = 0.1$.

Usage

```
riem.rmml(riemobj, label, lambda = 0.1, as.dist = FALSE)
```

Arguments

`riemobj`	a S3 `"riemdata"` class for N manifold-valued data.
`label`	a length-N vector of class labels. `NA` values are omitted.
`lambda`	regularization parameter. If $\lambda \leq 0$, no regularization is applied.
`as.dist`	logical; if `TRUE`, it returns `dist` object, else it returns a symmetric matrix.

Value

a S3 `dist` object or $(N \times N)$ symmetric matrix of pairwise distances according to `as.dist` parameter.

References

Zhu P, Cheng H, Hu Q, Wang Q, Zhang C (2018). "Towards Generalized and Efficient Metric Learning on Riemannian Manifold." In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, 3235–3241. ISBN 978-0-9992411-2-7, doi: 10.24963/ijcai.2018/449.

Examples

```
#------------------------------------------------------------------
#            Distance between Two Classes of SPD Matrices
#
#  Class 1 : Empirical Covariance from Standard Normal Distribution
#  Class 2 : Empirical Covariance from Perturbed 'iris' dataset
#------------------------------------------------------------------
## DATA GENERATION
data(iris)
ndata  = 10
mydata = list()
for (i in 1:ndata){
  mydata[[i]] = stats::cov(matrix(rnorm(100*4),ncol=4))
}
for (i in (ndata+1):(2*ndata)){
  tmpdata = as.matrix(iris[,1:4]) + matrix(rnorm(150*4,sd=0.5),ncol=4)
  mydata[[i]] = stats::cov(tmpdata)
}
myriem = wrap.spd(mydata)
mylabs = rep(c(1,2), each=ndata)

## COMPUTE GEODESIC AND RMML PAIRWISE DISTANCE
pdgeo = riem.pdist(myriem)
pdmdl = riem.rmml(myriem, label=mylabs)

## VISUALIZE
opar = par(no.readonly=TRUE)
par(mfrow=c(1,2), pty="s")
image(pdgeo[,(2*ndata):1], main="geodesic distance", axes=FALSE)
image(pdmdl[,(2*ndata):1], main="RMML distance", axes=FALSE)
par(opar)
```

`riem.seb` *Find the Smallest Enclosing Ball*

Description

Given N observations $X_1, X_2, \ldots, X_N \in \mathcal{M}$, find the smallest enclosing ball.

Usage

```
riem.seb(riemobj, method = c("aa2013"), maxiter = 50, eps = 1e-05)
```

Arguments

`riemobj`	a S3 `"riemdata"` class for N manifold-valued data.
`method`	(case-insensitive) name of the algorithm to be used as follows; `"aa2013"` Arnaudon and Nielsen (2013).

`maxiter`	maximum number of iterations to be run.
`eps`	tolerance level for stopping criterion.

Value

a named list containing

center a matrix on $\mathcal{M}$ that minimizes the radius.

radius the minimal radius with respect to the `center`.

References

Bâdoiu M, Clarkson KL (2003). "Smaller core-sets for balls." In *Proceedings of the fourteenth annual ACM-SIAM symposium on discrete algorithms*, SODA '03, 801–802. ISBN 0-89871-538-5.

Arnaudon M, Nielsen F (2013). "On approximating the Riemannian 1-center." *Computational Geometry*, **46**(1), 93–104. ISSN 09257721, doi: 10.1016/j.comgeo.2012.04.007.

Examples

```
#------------------------------------------------------------------
#       Euclidean Example : samples from Standard Normal in R^2
#------------------------------------------------------------------
## GENERATE 25 OBSERVATIONS FROM N(0,I)
ndata  = 25
mymats = array(0,c(ndata, 2))
mydata = list()
for (i in 1:ndata){
  mydata[[i]] = stats::rnorm(2)
  mymats[i,]  = mydata[[i]]
}
myriem = wrap.euclidean(mydata)

## COMPUTE
sebobj = riem.seb(myriem)
center = as.vector(sebobj$center)
radius = sebobj$radius

## VISUALIZE
#  1. prepare the circle for drawing
theta  = seq(from=0, to=2*pi, length.out=100)
coords = radius*cbind(cos(theta), sin(theta))
coords = coords + matrix(rep(center, each=100), ncol=2)

#  2. draw
opar <- par(no.readonly=TRUE)
par(pty="s")
plot(coords, type="l", lwd=2, col="red",
     main="Euclidean SEB", xlab="x", ylab="y")
points(mymats, pch=19)                          # data
points(center[1], center[2], pch=19, col="blue") # center
par(opar)
```

`riem.tsne` *t-distributed Stochastic Neighbor Embedding*

Description

Given N observations $X_1, X_2, \ldots, X_N \in \mathcal{M}$, t-SNE mimicks the pattern of probability distributions over pairs of manifold-valued objects on low-dimensional target embedding space by minimizing Kullback-Leibler divergence.

Usage

```
riem.tsne(riemobj, ndim = 2, geometry = c("intrinsic", "extrinsic"), ...)
```

Arguments

`riemobj`	a S3 `"riemdata"` class for N manifold-valued data.
`ndim`	an integer-valued target dimension.
`geometry`	(case-insensitive) name of geometry; either geodesic (`"intrinsic"`) or embedded (`"extrinsic"`) geometry.
`...`	extra parameters for `Rtsne` algorithm, such as perplexity, momentum, and others.

Value

a named list containing

embed an $(N \times ndim)$ matrix whose rows are embedded observations.

stress discrepancy between embedded and original distances as a measure of error.

See Also

`Rtsne`

Examples

```
#-----------------------------------------------------------------
#          Example on Sphere : a dataset with three types
#
# 10 perturbed data points near (1,0,0) on S^2 in R^3
# 10 perturbed data points near (0,1,0) on S^2 in R^3
# 10 perturbed data points near (0,0,1) on S^2 in R^3
#-----------------------------------------------------------------
## GENERATE DATA
mydata = list()
for (i in 1:20){
  tgt = c(1, stats::rnorm(2, sd=0.1))
  mydata[[i]] = tgt/sqrt(sum(tgt^2))
}
```

```
for (i in 21:40){
  tgt = c(rnorm(1,sd=0.1),1,rnorm(1,sd=0.1))
  mydata[[i]] = tgt/sqrt(sum(tgt^2))
}
for (i in 41:60){
  tgt = c(stats::rnorm(2, sd=0.1), 1)
  mydata[[i]] = tgt/sqrt(sum(tgt^2))
}
myriem = wrap.sphere(mydata)
mylabs = rep(c(1,2,3), each=20)

## RUN THE ALGORITHM IN TWO GEOMETRIES
mypx = 5
embed2int = riem.tsne(myriem, ndim=2, geometry="intrinsic", perplexity=mypx)
embed2ext = riem.tsne(myriem, ndim=2, geometry="extrinsic", perplexity=mypx)

## VISUALIZE
opar = par(no.readonly=TRUE)
par(mfrow=c(1,2), pty="s")
plot(embed2int$embed, main="intrinsic t-SNE", col=mylabs, pch=19)
plot(embed2ext$embed, main="extrinsic t-SNE", col=mylabs, pch=19)
par(opar)
```

`sphere.runif` *Generate Uniform Samples on Sphere*

Description

It generates n random samples from $\mathcal{S}^{p-1}$. For convenient usage of users, we provide a number of options in terms of the return type.

Usage

```
sphere.runif(n, p, type = c("list", "matrix", "riemdata"))
```

Arguments

`n`	number of samples to be generated.
`p`	original dimension (of the ambient space).
`type`	return type;
	`"list"` a length-n list of length-p vectors.
	`"matrix"` a $(n \times p)$ where rows are unit vectors.
	`"riemdata"` a S3 object. See `wrap.sphere` for more details (*Default*).

Value

an object from one of the above by `type` option.

References

Chikuse Y (2003). *Statistics on Special Manifolds*, volume 174 of *Lecture Notes in Statistics*. Springer New York, New York, NY. ISBN 978-0-387-00160-9 978-0-387-21540-2, doi: 10.1007/9780387215402.

See Also

wrap.sphere

Examples

```
#-----------------------------------------------------------------
#                       Draw Samples on Sphere
#
# Multiple return types on S^4 in R^5
#-----------------------------------------------------------------
dat.list = sphere.runif(n=10, p=5, type="list")
dat.matx = sphere.runif(n=10, p=5, type="matrix")
dat.riem = sphere.runif(n=10, p=5, type="riemdata")
```

---

sphere.utest *Test of Uniformity on Sphere*

Description

Given N observations $\left\{^{\{X_1,X_2,...,X_M}\right\}$ on $\mathcal{S}^{p-1}$, it tests whether the data is distributed uniformly on the sphere.

Usage

```
sphere.utest(spobj, method = c("Rayleigh", "RayleighM"))
```

Arguments

spobj	a S3 "riemdata" class for N Sphere-valued data.
method	(case-insensitive) name of the test method containing "Rayleigh" original Rayleigh statistic. "RayleighM" modified Rayleigh statistic with better order of error.

Value

a (list) object of S3 class htest containing:

statistic a test statistic.

p.value p-value under H_0.

alternative alternative hypothesis.

method name of the test.

data.name name(s) of provided sample data.

References

Chikuse Y (2003). *Statistics on Special Manifolds*, volume 174 of *Lecture Notes in Statistics*. Springer New York, New York, NY. ISBN 978-0-387-00160-9 978-0-387-21540-2, doi: 10.1007/9780387215402.

Mardia KV, Jupp PE (eds.) (1999). *Directional Statistics*, Wiley Series in Probability and Statistics. John Wiley \& Sons, Inc., Hoboken, NJ, USA. ISBN 978-0-470-31697-9 978-0-471-95333-3, doi: 10.1002/9780470316979.

See Also

`wrap.sphere`

Examples

```
#-----------------------------------------------------------------
#   Compare Rayleigh's original and modified versions of the test
#-----------------------------------------------------------------
#  Data Generation
myobj = sphere.runif(n=100, p=5, type="riemdata")

#  Compare 2 versions : Original vs Modified Rayleigh
sphere.utest(myobj, method="rayleigh")
sphere.utest(myobj, method="rayleighm")
```

`stiefel.runif` *Generate Uniform Samples on Stiefel Manifold*

Description

It generates n random samples from Stiefel manifold $St(k, p)$.

Usage

```
stiefel.runif(n, k, p, type = c("list", "array", "riemdata"))
```

Arguments

`n`	number of samples to be generated.
`k`	dimension of the frame.
`p`	original dimension (of the ambient space).
`type`	return type;
	`"list"` a length-n list of $(p \times k)$ basis of k-frames.
	`"array"` a $(p \times k \times n)$ 3D array whose slices are k-frame basis.
	`"riemdata"` a S3 object. See `wrap.stiefel` for more details.

Value

an object from one of the above by `type` option.

References

Chikuse Y (2003). *Statistics on Special Manifolds*, volume 174 of *Lecture Notes in Statistics*. Springer New York, New York, NY. ISBN 978-0-387-00160-9 978-0-387-21540-2, doi: 10.1007/9780387215402.

See Also

`wrap.stiefel`

Examples

```
#-----------------------------------------------------------------
#                 Draw Samples on Stiefel Manifold
#
# Try Different Return Types with 3 Observations of 5-frames in R^10
#-----------------------------------------------------------------
#  GENERATION
dat.list = stiefel.runif(n=3, k=5, p=10, type="list")
dat.arr3 = stiefel.runif(n=3, k=5, p=10, type="array")
dat.riem = stiefel.runif(n=3, k=5, p=10, type="riemdata")
```

`stiefel.utest` *Test of Uniformity on Stiefel Manifold*

Description

Given the data on Stiefel manifold $St(k,p)$, it tests whether the data is distributed uniformly.

Usage

```
stiefel.utest(stobj, method = c("Rayleigh", "RayleighM"))
```

Arguments

`stobj`	a S3 `"riemdata"` class for N Stiefel-valued data.
`method`	(case-insensitive) name of the test method containing
	`"Rayleigh"` original Rayleigh statistic.
	`"RayleighM"` modified Rayleigh statistic with better order of error.

Value

a (list) object of S3 class htest containing:

statistic a test statistic.

p.value p-value under H_0.

alternative alternative hypothesis.

method name of the test.

data.name name(s) of provided sample data.

References

Chikuse Y (2003). *Statistics on Special Manifolds*, volume 174 of *Lecture Notes in Statistics*. Springer New York, New York, NY. ISBN 978-0-387-00160-9 978-0-387-21540-2, doi: 10.1007/9780387215402.

Mardia KV, Jupp PE (eds.) (1999). *Directional Statistics*, Wiley Series in Probability and Statistics. John Wiley \& Sons, Inc., Hoboken, NJ, USA. ISBN 978-0-470-31697-9 978-0-471-95333-3, doi: 10.1002/9780470316979.

See Also

wrap.stiefel

Examples

```
#---------------------------------------------------------------------
#   Compare Rayleigh's original and modified versions of the test
#
# Test 1. sample uniformly from St(2,4)
# Test 2. use perturbed principal components from 'iris' data in R^4
#         which is concentrated around a point to reject H0.
#---------------------------------------------------------------------
## DATA GENERATION
#  1. uniform data
myobj1 = stiefel.runif(n=100, k=2, p=4)

#  2. perturbed principal components
data(iris)
irdat = list()
for (n in 1:100){
   tmpdata    = iris[1:50,1:4] + matrix(rnorm(50*4,sd=0.5),ncol=4)
   irdat[[n]] = eigen(cov(tmpdata))$vectors[,1:2]
}
myobj2 = wrap.stiefel(irdat)

## TEST
#  1. uniform data
stiefel.utest(myobj1, method="Rayleigh")
stiefel.utest(myobj1, method="RayleighM")

#  2. concentrated data
```

```
stiefel.utest(myobj2, method="rayleIgh")   # method names are
stiefel.utest(myobj2, method="raYleiGhM")  # CASE - INSENSITIVE !
```

`wrap.correlation` *Prepare Data on Correlation Manifold*

Description

The collection of correlation matrices is considered as a subset (and quotient) of the well-known SPD manifold. In our package, it is defined as

$$\mathcal{C}_{++}^{p} = \{X \in \mathbf{R}^{p\times p} \mid X^{\top} = X,\ \text{rank}(X) = p,\ \text{diag}(X) = 1\}$$

where the rank condition means it is strictly positive definite. Please note that the geometry involving semi-definite correlation matrices is not the objective here.

Usage

```
wrap.correlation(input)
```

Arguments

`input`	correlation data matrices to be wrapped as `riemdata` class. Following inputs are considered,
	array an $(p \times p \times n)$ array where each slice along 3rd dimension is a correlation matrix.
	list a length-n list whose elements are $(p \times p)$ correlation matrices.

Value

a named `riemdata` S3 object containing

data a list of $(p \times p)$ correlation matrices.

size size of each correlation matrix.

name name of the manifold of interests, *"correlation"*

Examples

```
#-----------------------------------------------------------------
#                 Checker for Two Types of Inputs
#
#  5 observations; empirical correlation of normal observations.
#-----------------------------------------------------------------
#  Data Generation
d1 = array(0,c(3,3,5))
d2 = list()
for (i in 1:5){
```

```
  dat = matrix(rnorm(10*3),ncol=3)
  d1[,,i] = stats::cor(dat)
  d2[[i]] = d1[,,i]
}

#  Run
test1 = wrap.correlation(d1)
test2 = wrap.correlation(d2)
```

`wrap.euclidean` *Prepare Data on Euclidean Space*

Description

Euclidean space $\mathbf{R}^p$ is the most common space for data analysis, which can be considered as a Riemannian manifold with flat metric. Since the space of matrices is isomorphic to Euclidean space after vectorization, we consider the inputs as p-dimensional vectors.

Usage

```
wrap.euclidean(input)
```

Arguments

`input`	data vectors to be wrapped as `riemdata` class. Following inputs are considered, **matrix** an $(n \times p)$ matrix of row observations. **list** a length-n list whose elements are length-p vectors.

Value

a named `riemdata` S3 object containing

data a list of $(p \times 1)$ matrices in $\mathbf{R}^p$.

size dimension of the ambient space.

name name of the manifold of interests, *"euclidean"*

Examples

```
#------------------------------------------------------------------
#                 Checker for Two Types of Inputs
#
#  Generate 5 observations in R^3 in Matrix and List.
#------------------------------------------------------------------
## DATA GENERATION
d1 = array(0,c(5,3))
d2 = list()
for (i in 1:5){
```

```
  single  = stats::rnorm(3)
  d1[i,]  = single
  d2[[i]] = single
}

## RUN
test1 = wrap.euclidean(d1)
test2 = wrap.euclidean(d2)
```

`wrap.grassmann` *Prepare Data on Grassmann Manifold*

Description

Grassmann manifold $Gr(k,p)$ is the set of k-planes, or k-dimensional subspaces in R^p, which means that for a given matrix $Y \in \mathbf{R}p \times k$, the column space $SPAN(Y)$ is an element in Grassmann manifold. We use a convention that each element in $Gr(k,p)$ is represented as an orthonormal basis (ONB) $X \in \mathbf{R}^{p \times k}$ where

$$X^\top X = I_k.$$

If not provided in such a form, this wrapper takes a QR decomposition of the given data to recover a corresponding ONB.

Usage

```
wrap.grassmann(input)
```

Arguments

`input`	data matrices to be wrapped as `riemdata` class. Following inputs are considered,
	array an $(p \times k \times n)$ array where each slice along 3rd dimension is a k-subspace basis in dimension p.
	list a length-n list whose elements are $(p \times k)$ basis for k-subspace.

Value

a named `riemdata` S3 object containing

data a list of k-subspace basis matrices.

size size of each k-subspace basis matrix.

name name of the manifold of interests, *"grassmann"*

Examples

```
#------------------------------------------------------------------
#                 Checker for Two Types of Inputs
#
#  Generate 5 observations in Gr(2,4)
#------------------------------------------------------------------
#  Generation
d1 = array(0,c(4,2,5))
d2 = list()
for (i in 1:5){
  d1[,,i] = matrix(rnorm(4*2), ncol=2)
  d2[[i]] = d1[,,i]
}

#  Run
test1 = wrap.grassmann(d1)
test2 = wrap.grassmann(d2)
```

`wrap.multinomial` *Prepare Data on Multinomial Manifold*

Description

Multinomial manifold is referred to the data that is nonnegative and sums to 1. Also known as probability simplex or positive orthant, we denote $(p-1)$ simplex in $\mathbf{R}^p$ by

$$\Delta^{p-1} = \{x \in \mathbf{R}^p \mid \sum_{i=1}^{p} x_i = 1, x_i > 0\}$$

in that data are positive L_1 unit-norm vectors. In `wrap.multinomial`, normalization is applied when each data point is not on the simplex, but if vectors contain values not in $(0, 1)$, it returns errors.

Usage

```
wrap.multinomial(input)
```

Arguments

`input`	data vectors to be wrapped as `riemdata` class. Following inputs are considered,
	matrix an $(n \times p)$ matrix of row observations.
	list a length-n list whose elements are length-p vectors.

Value

a named `riemdata` S3 object containing

data a list of $(p \times 1)$ matrices in Δ^{p-1}.

size dimension of the ambient space.

name name of the manifold of interests, *"multinomial"*

Examples

```
#-----------------------------------------------------------------
#                 Checker for Two Types of Inputs
#-----------------------------------------------------------------
## DATA GENERATION
d1 = array(0,c(5,3))
d2 = list()
for (i in 1:5){
  single  = abs(stats::rnorm(3))
  d1[i,]  = single
  d2[[i]] = single
}

## RUN
test1 = wrap.multinomial(d1)
test2 = wrap.multinomial(d2)
```

`wrap.rotation` *Prepare Data on Rotation Group*

Description

Rotation group, also known as special orthogonal group, is a Riemannian manifold

$$SO(p) = \{Q \in \mathbf{R}^{p \times p} \mid Q^\top Q = I, \det(Q) = 1\}$$

where the name originates from an observation that when $p = 2, 3$ these matrices are rotation of shapes/configurations.

Usage

```
wrap.rotation(input)
```

Arguments

`input`	data matrices to be wrapped as `riemdata` class. Following inputs are considered,
	array a $(p \times p \times n)$ array where each slice along 3rd dimension is a rotation matrix.
	list a length-n list whose elements are $(p \times p)$ rotation matrices.

Value

a named `riemdata` S3 object containing

data a list of $(p \times p)$ rotation matrices.

size size of each rotation matrix.

name name of the manifold of interests, *"rotation"*

Examples

```
#-------------------------------------------------------------------
#                 Checker for Two Types of Inputs
#-------------------------------------------------------------------
## DATA GENERATION
d1 = array(0,c(3,3,5))
d2 = list()
for (i in 1:5){
  single  = qr.Q(qr(matrix(rnorm(9),nrow=3)))
  d1[,,i] = single
  d2[[i]] = single
}

## RUN
test1 = wrap.rotation(d1)
test2 = wrap.rotation(d2)
```

`wrap.spd` *Prepare Data on Symmetric Positive-Definite (SPD) Manifold*

Description

The collection of symmetric positive-definite matrices is a well-known example of matrix manifold. It is defined as

$$\mathcal{S}_{++}^{p} = \{X \in \mathbf{R}^{p \times p} \mid X^{\top} = X,\ \text{rank}(X) = p\}$$

where the rank condition means it is strictly positive definite. Please note that the geometry involving semi-definite matrices is considered in `wrap.spdk`.

Usage

```
wrap.spd(input)
```

Arguments

`input`	SPD data matrices to be wrapped as `riemdata` class. Following inputs are considered,
	array an $(p \times p \times n)$ array where each slice along 3rd dimension is a SPD matrix.
	list a length-n list whose elements are $(p \times p)$ SPD matrices.

Value

a named `riemdata` S3 object containing

data a list of $(p \times p)$ correlation matrices.

size size of each correlation matrix.

name name of the manifold of interests, *"spd"*

Examples

```
#-----------------------------------------------------------------
#                 Checker for Two Types of Inputs
#
#  Generate 5 observations; empirical covariance of normal observations.
#-----------------------------------------------------------------
#  Data Generation
d1 = array(0,c(3,3,5))
d2 = list()
for (i in 1:5){
  dat = matrix(rnorm(10*3),ncol=3)
  d1[,,i] = stats::cov(dat)
  d2[[i]] = d1[,,i]
}

#  Run
test1 = wrap.spd(d1)
test2 = wrap.spd(d2)
```

`wrap.spdk` *Prepare Data on SPD Manifold of Fixed-Rank*

Description

When $(p \times p)$ SPD matrices are of fixed-rank $k < p$, they form a geometric structure represented by $(p \times k)$ matrices,

$$SPD(k,p) = \{X \in \mathbf{R}^{(p \times p)} \mid YY^\top = X, \text{rank}(X) = k\}$$

It's key difference from $\mathcal{S}_{++}^p$ is that all matrices should be of fixed rank k where k is usually smaller than p. Inputs are given as $(p \times p)$ matrices with specified k and `wrap.spdk` automatically decomposes input square matrices into rank-k representation matrices.

Usage

```
wrap.spdk(input, k)
```

Arguments

`input`	data matrices to be wrapped as `riemdata` class. Following inputs are considered,
	array a $(p \times p \times n)$ array where each slice along 3rd dimension is a rank-k matrix.
	list a length-n list whose elements are $(p \times p)$ matrices of rank-k.
`k`	rank of the SPD matrices.

Value

a named `riemdata` S3 object containing

data a list of $(p \times k)$ representation of the corresponding rank-k SPSD matrices.

size size of each representation matrix.

name name of the manifold of interests, *"spdk"*

References

Journée M, Bach F, Absil P, Sepulchre R (2010). "Low-rank optimization on the cone of positive semidefinite matrices." *SIAM Journal on Optimization*, **20**(5), 2327–2351.

Examples

```
#-------------------------------------------------------------------
#                 Checker for Two Types of Inputs
#-------------------------------------------------------------------
#  Data Generation
d1 = array(0,c(10,10,3))
d2 = list()
for (i in 1:3){
  dat = matrix(rnorm(10*10),ncol=10)
  d1[,,i] = stats::cov(dat)
  d2[[i]] = d1[,,i]
}

#  Run
test1 = wrap.spdk(d1, k=2)
test2 = wrap.spdk(d2, k=2)
```

`wrap.sphere` *Prepare Data on Sphere*

Description

The unit hypersphere (sphere, for short) is one of the most fundamental curved space in studying geometry. Precisely, we denote $(p-1)$ sphere in $\mathbf{R}^p$ by

$$\mathcal{S}^{p-1} = \{x \in \mathbf{R}^p \mid x^\top x = \|x\|^2 = 1\}$$

where vectors are of unit norm. In `wrap.sphere`, normalization is applied when each data point is not on the unit sphere.

Usage

```
wrap.sphere(input)
```

Arguments

`input`	data vectors to be wrapped as `riemdata` class. Following inputs are considered,
	matrix an $(n \times p)$ matrix of row observations of unit norm.
	list a length-n list whose elements are length-p vectors of unit norm.

Value

a named `riemdata` S3 object containing

data a list of $(p \times 1)$ matrices in $\mathcal{S}^{p-1}$.

size dimension of the ambient space.

name name of the manifold of interests, *"sphere"*

Examples

```
#-----------------------------------------------------------------
#                 Checker for Two Types of Inputs
#
#  Generate 5 observations in S^2 embedded in R^3.
#-----------------------------------------------------------------
## DATA GENERATION
d1 = array(0,c(5,3))
d2 = list()
for (i in 1:5){
  single  = stats::rnorm(3)
  d1[i,]  = single
  d2[[i]] = single
}

## RUN
test1 = wrap.sphere(d1)
test2 = wrap.sphere(d2)
```

`wrap.stiefel` *Prepare Data on (Compact) Stiefel Manifold*

Description

Stiefel manifold $St(k,p)$ is the set of k-frames in $\mathbf{R}^p$, which is indeed a Riemannian manifold. For usage in **Riemann** package, each data point is represented as a matrix by the convention

$$St(k,p) = \{X \in \mathbf{R}^{p\times k} \mid X^\top X = I_k\}$$

which means that columns are orthonormal. When the provided matrix is not an orthonormal basis as above, `wrap.stiefel` applies orthogonalization to extract valid basis information.

Usage

```
wrap.stiefel(input)
```

Arguments

`input`	data matrices to be wrapped as `riemdata` class. Following inputs are considered, **array** a $(p \times k \times n)$ array where each slice along 3rd dimension is a k-frame. **list** a length-n list whose elements are $(p \times k)$ k-frames.

Value

a named `riemdata` S3 object containing

data a list of k-frame orthonormal matrices.

size size of each k-frame basis matrix.

name name of the manifold of interests, *"stiefel"*

Examples

```
#-----------------------------------------------------------------
#                 Checker for Two Types of Inputs
#
#  Generate 5 observations in St(2,4)
#-----------------------------------------------------------------
#  Data Generation by QR Decomposition
d1 = array(0,c(4,2,5))
d2 = list()
for (i in 1:5){
  d1[,,i] = qr.Q(qr(matrix(rnorm(4*2),ncol=2)))
  d2[[i]] = d1[,,i]
}

#  Run
test1 = wrap.stiefel(d1)
test2 = wrap.stiefel(d2)
```

Index