Package: Riemann (via r-universe)

October 13, 2026

Type Package

Title Learning with Data on Riemannian Manifolds

Version 0.1.7

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, Rdimtools, T4cluster, T4transport, DEoptim, lpSolve, Matrix, maotai (>= 0.2.2), stats, utils

Encoding UTF-8

URL https://www.kisungyou.com/Riemann/

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

LinkingTo Rcpp, RcppArmadillo

RoxygenNote 7.3.3

RdMacros Rdpack

Depends R (>= 2.10)

Suggests testthat (>= 3.0.0)

LazyData true

Config/testthat/edition 3

NeedsCompilation yes

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

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

Repository https://cran.r-universe.dev

Date/Publication 2026-03-25 05:10:02 UTC

RemoteUrl https://github.com/cran/Riemann

RemoteRef HEAD

RemoteSha 737dd6f09b8a93402306fce14ec481d51734c647

Contents

acg . . . 3
cities . . . 5
density . . . 6
ERP . . . 7
gorilla . . . 7
grassmann.optmacg . . . 8
grassmann.runif . . . 10
grassmann.utest . . . 11
hands . . . 12
label . . . 13
loglkd . . . 14
macg . . . 15
moSL . . . 17
moSN . . . 19
orbital . . . 21
passiflora . . . 22
predict.m2skreg . . . 23
riem.clrq . . . 24
riem.coreset18B . . . 26
riem.distlp . . . 28
riem.dtw . . . 30
riem.fanova . . . 31
riem.hclust . . . 33
riem.interp . . . 34
riem.interps . . . 35
riem.isomap . . . 37
riem.kmeans . . . 38
riem.kmeans18B . . . 40
riem.kmeanspp . . . 42
riem.kmedoids . . . 44
riem.knn . . . 45
riem.kpca . . . 47
riem.m2skreg . . . 48
riem.m2skregCV . . . 50
riem.mds . . . 51
riem.mean . . . 53
riem.median . . . 54
riem.nmshift . . . 56
riem.pdist . . . 57
riem.pdist2 . . . 59
riem.pga . . . 60

riem.phate . . . 61
riem.rmml . . . 63
riem.sammon . . . 64
riem.sc05Z . . . 66
riem.scNJW . . . 67
riem.scSM . . . 69
riem.scUL . . . 71
riem.seb . . . 72
riem.test2bg14 . . . 74
riem.test2wass . . . 76
riem.tsne . . . 78
riem.wasserstein . . . 79
rmvnorm . . . 81
spd.geometry . . . 82
spd.pdist . . . 83
spd.wassbary . . . 84
sphere.convert . . . 85
sphere.runif . . . 86
sphere.utest . . . 87
splaplace . . . 88
spnorm . . . 90
stiefel.optSA . . . 92
stiefel.runif . . . 93
stiefel.utest . . . 95
wrap.correlation . . . 96
wrap.euclidean . . . 97
wrap.grassmann . . . 98
wrap.landmark . . . 99
wrap.multinomial . . . 100
wrap.rotation . . . 101
wrap.spd . . . 102
wrap.spdk . . . 103
wrap.sphere . . . 105
wrap.stiefel . . . 106

Index **108**

---

acg *Angular Central Gaussian Distribution*

Description

For a hypersphere $\mathcal{S}^{p-1}$ in $\mathbf{R}^p$, Angular Central Gaussian (ACG) distribution $ACG_p(A)$ is defined via a density

$$f(x|A) = |A|^{-1/2}(x^\top A^{-1}x)^{-p/2}$$

with respect to the uniform measure on $\mathcal{S}^{p-1}$ and A is a symmetric positive-definite matrix. Since $f(x|A) = f(-x|A)$, it can also be used as an axial distribution on real projective space, which is

unit sphere modulo $\{+1, -1\}$. One constraint we follow is that $f(x|A) = f(x|cA)$ for $c > 0$ in that we use a normalized version for numerical stability by restricting $tr(A) = p$.

Usage

```
dacg(datalist, A)

racg(n, A)

mle.acg(datalist, ...)
```

Arguments

`datalist`	a list of length-p unit-norm vectors.
`A`	a $(p \times p)$ symmetric positive-definite matrix.
`n`	the number of samples to be generated.
`...`	extra parameters for computations, including
	maxiter maximum number of iterations to be run (default:50).
	eps tolerance level for stopping criterion (default: 1e-5).

Value

`dacg` gives a vector of evaluated densities given samples. `racg` generates unit-norm vectors in $\mathbf{R}^p$ wrapped in a list. `mle.acg` estimates the SPD matrix A.

References

Tyler DE (1987). "Statistical analysis for the angular central Gaussian distribution on the sphere." *Biometrika*, **74**(3), 579–589. ISSN 0006-3444, 1464-3510.

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

Examples

```
# ------------------------------------------------------------------
#          Example with Angular Central Gaussian Distribution
#
# Given a fixed A, generate samples and estimate A via ML.
# ------------------------------------------------------------------
## GENERATE AND MLE in R^5
#  Generate data
Atrue = diag(5)          # true SPD matrix
sam1  = racg(50,  Atrue) # random samples
sam2  = racg(100, Atrue)

#  MLE
Amle1 = mle.acg(sam1)
Amle2 = mle.acg(sam2)
```

```
#  Visualize
opar <- par(no.readonly=TRUE)
par(mfrow=c(1,3), pty="s")
image(Atrue[,5:1], axes=FALSE, main="true SPD")
image(Amle1[,5:1], axes=FALSE, main="MLE with n=50")
image(Amle2[,5:1], axes=FALSE, main="MLE with n=100")
par(opar)
```

`cities` *Data : Populated Cities in the U.S.*

Description

As of January 2006, there are 60 cities in the contiguous U.S. with population size larger than 300000. We extracted information of the cities from the data delivered by **maps** package. Variables `coord` and `cartesian` are two identical representations of locations, which can be mutually converted by `sphere.convert`.

Usage

```
data(cities)
```

Format

a named list containing

names a length-60 vector of city names.

coord a (60×2) matrix of latitude and longitude.

cartesian a (60×3) matrix of cartesian coordinates on the unit sphere.

population a length-60 vector of cities' populations.

See Also

`wrap.sphere`

Examples

```
## LOAD THE DATA AND WRAP AS RIEMOBJ
data(cities)
myriem = wrap.sphere(cities$cartesian)

## COMPUTE INTRINSIC/EXTRINSIC MEANS
intmean = as.vector(riem.mean(myriem, geometry="intrinsic")$mean)
extmean = as.vector(riem.mean(myriem, geometry="extrinsic")$mean)

## CONVERT TO GEOGRAPHIC COORDINATES (Lat/Lon)
geo.int = sphere.xyz2geo(intmean)
```

```
geo.ext = sphere.xyz2geo(extmean)
```

`density` *S3 method for mixture model : evaluate density*

Description

Compute density for a fitted mixture model.

Usage

```
density(object, newdata)
```

Arguments

`object`	a fitted mixture model of `riemmix` class.
`newdata`	data of n objects (vectors, matrices) that can be wrapped by one of `wrap.*` functions in the **Riemann** package.

Value

a length-n vector of class labels.

Examples

```
# ------------------------------------------------- #
#            FIT A MODEL & APPLY THE METHOD
# ------------------------------------------------- #
# Load the 'city' data and wrap as 'riemobj'
data(cities)
locations = cities$cartesian
embed2    = array(0,c(60,2))
for (i in 1:60){
   embed2[i,] = sphere.xyz2geo(locations[i,])
}

# Fit a model
k3 = moSN(locations, k=3)

# Evaluate
newdensity = density(k3, locations)
```

ERP *Data : EEG Covariances for Event-Related Potentials*

Description

This dataset delivers 216 covariance matrices from EEG ERPs with 4 different known classes by types of sources. Among 60 channels, only 32 channels are taken and sample covariance matrix is computed for each participant. The data is taken from a Python library mne's sample data.

Usage

```
data(ERP)
```

Format

a named list containing

covariance an $(32 \times 32 \times 216)$ array of covariance matrices.

label a length-216 factor of 4 different classes.

See Also

`wrap.spd`

Examples

```
## LOAD THE DATA AND WRAP AS RIEMOBJ
data(ERP)
myriem = wrap.spd(ERP$covariance)
```

gorilla *Data : Gorilla Skull*

Description

This is 29 male and 30 female gorillas' skull landmark data where each individual is represented as 8-ad/landmarks in 2 dimensions. This is a re-arranged version of the data from **shapes** package.

Usage

```
data(gorilla)
```

Format

a named list containing

male a 3d array of size $(8 \times 2 \times 29)$

female a 3d array of size $(8 \times 2 \times 30)$

References

Dryden IL, Mardia KV (2016). *Statistical shape analysis with applications in R*, Wiley series in probability and statistics, Second edition edition. John Wiley and Sons, Chichester, UK ; Hoboken, NJ. ISBN 978-1-119-07251-5 978-1-119-07250-8.

Reno PL, Meindl RS, McCollum MA, Lovejoy CO (2003). "Sexual dimorphism in Australopithecus afarensis was similar to that of modern humans." *Proceedings of the National Academy of Sciences*, 100(16):9404–9409.

See Also

wrap.landmark

Examples

```
data(gorilla)                                # load the data
riem.female = wrap.landmark(gorilla$female) # wrap as RIEMOBJ
opar <- par(no.readonly=TRUE)
for (i in 1:30){
  if (i < 2){
    plot(riem.female$data[[i]], cex=0.5,
         xlim=c(-1,1)/2, ylim=c(-2,2)/5,
         main="30 female gorilla skull preshapes",
         xlab="dimension 1", ylab="dimension 2")
    lines(riem.female$data[[i]])
  } else {
    points(riem.female$data[[i]], cex=0.5)
    lines(riem.female$data[[i]])
  }
}
par(opar)
```

---

grassmann.optmacg *Estimation of Distribution Algorithm with MACG Distribution*

Description

For a function $f : Gr(k, p) \to \mathbf{R}$, find the minimizer and the attained minimum value with estimation of distribution algorithm using MACG distribution.

Usage

```
grassmann.optmacg(func, p, k, ...)
```

Arguments

`func`	a function to be *minimized*.
`p`	dimension parameter as in $Gr(k, p)$.
`k`	dimension parameter as in $Gr(k, p)$.
`...`	extra parameters including

n.start number of runs; algorithm is executed `n.start` times (default: 10).

maxiter maximum number of iterations for each run (default: 100).

popsize the number of samples generated at each step for stochastic search (default: 100).

ratio ratio in $(0, 1)$ where top `ratio*popsize` samples are chosen for parameter update (default: 0.25).

print.progress a logical; if `TRUE`, it prints each iteration (default: `FALSE`).

Value

a named list containing:

cost minimized function value.

solution a $(p \times k)$ matrix that attains the `cost`.

Examples

```
#-----------------------------------------------------------------
#               Optimization for Eigen-Decomposition
#
# Given (5x5) covariance matrix S, eigendecomposition is can be
# considered as an optimization on Grassmann manifold. Here,
# we are trying to find top 3 eigenvalues and compare.
#-----------------------------------------------------------------

## PREPARE
A = cov(matrix(rnorm(100*5), ncol=5)) # define covariance
myfunc <- function(p){                # cost function to minimize
  return(sum(-diag(t(p)%*%A%*%p)))
}

## SOLVE THE OPTIMIZATION PROBLEM
Aout = grassmann.optmacg(myfunc, p=5, k=3, popsize=100, n.start=30)

## COMPUTE EIGENVALUES
#  1. USE SOLUTIONS TO THE ABOVE OPTIMIZATION
abase   = Aout$solution
eig3sol = sort(diag(t(abase)%*%A%*%abase), decreasing=TRUE)

#  2. USE BASIC 'EIGEN' FUNCTION
```

```
eig3dec = sort(eigen(A)$values, decreasing=TRUE)[1:3]

## VISUALIZE
opar <- par(no.readonly=TRUE)
yran = c(min(min(eig3sol),min(eig3dec))*0.95,
         max(max(eig3sol),max(eig3dec))*1.05)
plot(1:3, eig3sol, type="b", col="red",  pch=19, ylim=yran,
     xlab="index", ylab="eigenvalue", main="compare top 3 eigenvalues")
lines(1:3, eig3dec, type="b", col="blue", pch=19)
legend(1.55, max(yran), legend=c("optimization","decomposition"), col=c("red","blue"),
       lty=rep(1,2), pch=19)
par(opar)
```

`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.

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.

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

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 !
```

---

hands *Data : Left Hands*

Description

This dataset contains 10 shapes of 4 subjects's left hands where each shape is represented by 56 landmark points. For each person, first six shapes are equally spaced sequence from maximally to minimally spread fingures. The rest are arbitrarily chosen with two constraints; (1) the palm should face the support and (2) the contour should contain no crossins.

Usage

```
data(hands)
```

Format

a named list containing

data an $(56 \times 2 \times 40)$ array of landmarks for 40 subjects.

person a length-40 vector of subject indices.

References

Stegmann M, Gomez D (2002) "A Brief Introduction to Statistical Shape Analysis." *Informatics and Mathematical Modelling, Technical University of Denmark, DTU.*

See Also

`wrap.landmark`

Examples

```
## LOAD THE DATA
data(hands)

## VISUALIZE 6 HANDS OF PERSON 1
opar <- par(no.readonly=TRUE)
par(mfrow=c(2,3))
for (i in 1:6){
  xx = hands$data[,1,i]
  yy = hands$data[,2,i]
  plot(xx,yy,"b", cex=0.9)
}
par(opar)
```

---

label *S3 method for mixture model : predict labels*

Description

Given a fitted mixture model of K components, predict labels of observations accordingly.

Usage

```
label(object, newdata)
```

Arguments

`object`	a fitted mixture model of `riemmix` class.
`newdata`	data of n objects (vectors, matrices) that can be wrapped by one of `wrap.*` functions in the **Riemann** package.

Value

a length-n vector of class labels.

Examples

```
# ------------------------------------------------- #
#            FIT A MODEL & APPLY THE METHOD
# ------------------------------------------------- #
# Load the 'city' data and wrap as 'riemobj'
data(cities)
locations = cities$cartesian
embed2    = array(0,c(60,2))
for (i in 1:60){
   embed2[i,] = sphere.xyz2geo(locations[i,])
}

# Fit a model
k3 = moSN(locations, k=3)

# Evaluate
newlabel = label(k3, locations)
```

`loglkd` *S3 method for mixture model : log-likelihood*

Description

Given a fitted mixture model $f(x)$ and observations $x_1, \ldots, x_n \in \mathcal{M}$, compute the log-likelihood

$$L = \log \prod_{i=1}^{n} f(x_i) = \sum_{i=1}^{n} \log f(x_i)$$

.

Usage

```
loglkd(object, newdata)
```

Arguments

`object`	a fitted mixture model of `riemmix` class.
`newdata`	data of n objects (vectors, matrices) that can be wrapped by one of `wrap.*` functions in the **Riemann** package.

Value

the log-likelihood.

Examples

```
# ------------------------------------------------------ #
#            FIT A MODEL & APPLY THE METHOD
# ------------------------------------------------------ #
# Load the 'city' data and wrap as 'riemobj'
data(cities)
locations = cities$cartesian
embed2    = array(0,c(60,2))
for (i in 1:60){
   embed2[i,] = sphere.xyz2geo(locations[i,])
}

# Fit a model
k3 = moSN(locations, k=3)

# Evaluate
newloglkd = round(loglkd(k3, locations), 3)
print(paste0("Log-likelihood for K=3 model fit : ", newloglkd))
```

---

macg *Matrix Angular Central Gaussian Distribution*

Description

For Stiefel and Grassmann manifolds $St(r,p)$ and $Gr(r,p)$, the matrix variant of ACG distribution is known as Matrix Angular Central Gaussian (MACG) distribution $MACG_{p,r}(\Sigma)$ with density

$$f(X|\Sigma) = |\Sigma|^{-r/2}|X^\top \Sigma^{-1} X|^{-p/2}$$

where Σ is a $(p \times p)$ symmetric positive-definite matrix. Similar to vector-variate ACG case, we follow a convention that $tr(\Sigma) = p$.

Usage

```
dmacg(datalist, Sigma)

rmacg(n, r, Sigma)

mle.macg(datalist, ...)
```

Arguments

datalist	a list of $(p \times r)$ orthonormal matrices.
Sigma	a $(p \times p)$ symmetric positive-definite matrix.
n	the number of samples to be generated.
r	the number of basis.

... extra parameters for computations, including

maxiter maximum number of iterations to be run (default:50).

eps tolerance level for stopping criterion (default: 1e-5).

Value

`dmacg` gives a vector of evaluated densities given samples. `rmacg` generates $(p \times r)$ orthonormal matrices wrapped in a list. `mle.macg` estimates the SPD matrix Σ.

References

Chikuse Y (1990). "The matrix angular central Gaussian distribution." *Journal of Multivariate Analysis*, **33**(2), 265–274. ISSN 0047259X.

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

Kent JT, Ganeiber AM, Mardia KV (2013). "A new method to simulate the Bingham and related distributions in directional data analysis with applications." *arXiv:1310.8110*.

See Also

acg

Examples

```
# ---------------------------------------------------------------------
#          Example with Matrix Angular Central Gaussian Distribution
#
# Given a fixed Sigma, generate samples and estimate Sigma via ML.
# ---------------------------------------------------------------------
## GENERATE AND MLE in St(2,5)/Gr(2,5)
#  Generate data
Strue = diag(5)                  # true SPD matrix
sam1  = rmacg(n=50,  r=2, Strue) # random samples
sam2  = rmacg(n=100, r=2, Strue) # random samples

#  MLE
Smle1 = mle.macg(sam1)
Smle2 = mle.macg(sam2)

#  Visualize
opar <- par(no.readonly=TRUE)
par(mfrow=c(1,3), pty="s")
image(Strue[,5:1], axes=FALSE, main="true SPD")
image(Smle1[,5:1], axes=FALSE, main="MLE with n=50")
image(Smle2[,5:1], axes=FALSE, main="MLE with n=100")
par(opar)
```

moSL *Finite Mixture of Spherical Laplace Distributions*

Description

For n observations on a $(p-1)$ sphere in $\mathbf{R}^p$, a finite mixture model is fitted whose components are spherical Laplace distributions via the following model

$$f(x; \{w_k, \mu_k, \sigma_k\}_{k=1}^K) = \sum_{k=1}^K w_k SL(x; \mu_k, \sigma_k)$$

with parameters w_k's for component weights, μ_k's for component locations, and σ_k's for component scales.

Usage

```
moSL(
  data,
  k = 2,
  same.sigma = FALSE,
  variants = c("soft", "hard", "stochastic"),
  ...
)

## S3 method for class 'moSL'
loglkd(object, newdata)

## S3 method for class 'moSL'
label(object, newdata)

## S3 method for class 'moSL'
density(object, newdata)
```

Arguments

`data`	data vectors in form of either an $(n \times p)$ matrix or a length-n list. See `wrap.sphere` for descriptions on supported input types.
`k`	the number of clusters (default: 2).
`same.sigma`	a logical; `TRUE` to use same scale parameter across all components, or `FALSE` otherwise.
`variants`	type of the class assignment methods, one of `"soft"`,`"hard"`, and `"stochastic"`.
`...`	extra parameters including **maxiter** the maximum number of iterations (default: 50). **eps** stopping criterion for the EM algorithm (default: 1e-6). **printer** a logical; `TRUE` to show history of the algorithm, `FALSE` otherwise.

`object`	a fitted `moSL` model from the `moSL` function.
`newdata`	data vectors in form of either an $(m \times p)$ matrix or a length-m list. See `wrap.sphere` for descriptions on supported input types.

Value

a named list of S3 class `riemmix` containing

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

loglkd log likelihood of the fitted model.

criteria a vector of information criteria.

parameters a list containing `proportion`, `location`, and `scale`. See the section for more details.

membership an $(n \times k)$ row-stochastic matrix of membership.

Parameters of the fitted model

A fitted model is characterized by three parameters. For k-mixture model on a $(p-1)$ sphere in $\mathbf{R}^p$, (1) `proportion` is a length-k vector of component weight that sums to 1, (2) `location` is an $(k \times p)$ matrix whose rows are per-cluster locations, and (3) `concentration` is a length-k vector of scale parameters for each component.

Note on S3 methods

There are three S3 methods; `loglkd`, `label`, and `density`. Given a random sample of size m as `newdata`, (1) `loglkd` returns a scalar value of the computed log-likelihood, (2) `label` returns a length-m vector of cluster assignments, and (3) `density` evaluates densities of every observation according ot the model fit.

Examples

```
# ------------------------------------------------------ #
#                 FITTING THE MODEL
# ------------------------------------------------------ #
# Load the 'city' data and wrap as 'riemobj'
data(cities)
locations = cities$cartesian
embed2    = array(0,c(60,2))
for (i in 1:60){
   embed2[i,] = sphere.xyz2geo(locations[i,])
}

# Fit the model with different numbers of clusters
k2 = moSL(locations, k=2)
k3 = moSL(locations, k=3)
k4 = moSL(locations, k=4)

# Visualize
opar <- par(no.readonly=TRUE)
par(mfrow=c(1,3))
plot(embed2, col=k2$cluster, pch=19, main="K=2")
```

```
plot(embed2, col=k3$cluster, pch=19, main="K=3")
plot(embed2, col=k4$cluster, pch=19, main="K=4")
par(opar)

# ------------------------------------------------- #
#                  USE S3 METHODS
# ------------------------------------------------- #
# Use the same 'locations' data as new data
# (1) log-likelihood
newloglkd = round(loglkd(k3, locations), 5)
fitloglkd = round(k3$loglkd, 5)
print(paste0("Log-likelihood for K=3 fitted    : ", fitloglkd))
print(paste0("Log-likelihood for K=3 predicted : ", newloglkd))

# (2) label
newlabel = label(k3, locations)

# (3) density
newdensity = density(k3, locations)
```

---

moSN *Finite Mixture of Spherical Normal Distributions*

Description

For n observations on a $(p-1)$ sphere in $\mathbf{R}^p$, a finite mixture model is fitted whose components are spherical normal distributions via the following model

$$f(x; \{w_k, \mu_k, \lambda_k\}_{k=1}^K) = \sum_{k=1}^K w_k SN(x; \mu_k, \lambda_k)$$

with parameters w_k's for component weights, μ_k's for component locations, and λ_k's for component concentrations.

Usage

```
moSN(
  data,
  k = 2,
  same.lambda = FALSE,
  variants = c("soft", "hard", "stochastic"),
  ...
)

## S3 method for class 'moSN'
loglkd(object, newdata)
```

```
## S3 method for class 'moSN'
label(object, newdata)

## S3 method for class 'moSN'
density(object, newdata)
```

Arguments

`data`	data vectors in form of either an $(n \times p)$ matrix or a length-n list. See `wrap.sphere` for descriptions on supported input types.
`k`	the number of clusters (default: 2).
`same.lambda`	a logical; `TRUE` to use same concentration parameter across all components, or `FALSE` otherwise.
`variants`	type of the class assignment methods, one of `"soft"`,`"hard"`, and `"stochastic"`.
`...`	extra parameters including **maxiter** the maximum number of iterations (default: 50). **eps** stopping criterion for the EM algorithm (default: 1e-6). **printer** a logical; `TRUE` to show history of the algorithm, `FALSE` otherwise.
`object`	a fitted `moSN` model from the `moSN` function.
`newdata`	data vectors in form of either an $(m \times p)$ matrix or a length-m list. See `wrap.sphere` for descriptions on supported input types.

Value

a named list of S3 class `riemmix` containing

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

loglkd log likelihood of the fitted model.

criteria a vector of information criteria.

parameters a list containing `proportion`, `center`, and `concentration`. See the section for more details.

membership an $(n \times k)$ row-stochastic matrix of membership.

Parameters of the fitted model

A fitted model is characterized by three parameters. For k-mixture model on a $(p-1)$ sphere in $\mathbf{R}^p$, (1) `proportion` is a length-k vector of component weight that sums to 1, (2) `center` is an $(k \times p)$ matrix whose rows are cluster centers, and (3) `concentration` is a length-k vector of concentration parameters for each component.

Note on S3 methods

There are three S3 methods; `loglkd`, `label`, and `density`. Given a random sample of size m as `newdata`, (1) `loglkd` returns a scalar value of the computed log-likelihood, (2) `label` returns a length-m vector of cluster assignments, and (3) `density` evaluates densities of every observation according ot the model fit.

References

You K, Suh C (2022). "Parameter Estimation and Model-Based Clustering with Spherical Normal Distribution on the Unit Hypersphere." *Computational Statistics and Data Analysis*, 107457. ISSN 01679473.

Examples

```
# ------------------------------------------------------ #
#                 FITTING THE MODEL
# ------------------------------------------------------ #
# Load the 'city' data and wrap as 'riemobj'
data(cities)
locations = cities$cartesian
embed2    = array(0,c(60,2))
for (i in 1:60){
   embed2[i,] = sphere.xyz2geo(locations[i,])
}

# Fit the model with different numbers of clusters
k2 = moSN(locations, k=2)
k3 = moSN(locations, k=3)
k4 = moSN(locations, k=4)

# Visualize
opar <- par(no.readonly=TRUE)
par(mfrow=c(1,3))
plot(embed2, col=k2$cluster, pch=19, main="K=2")
plot(embed2, col=k3$cluster, pch=19, main="K=3")
plot(embed2, col=k4$cluster, pch=19, main="K=4")
par(opar)

# ------------------------------------------------------ #
#                   USE S3 METHODS
# ------------------------------------------------------ #
# Use the same 'locations' data as new data
# (1) log-likelihood
newloglkd = round(loglkd(k3, locations), 3)
print(paste0("Log-likelihood for K=3 model fit : ", newloglkd))

# (2) label
newlabel = label(k3, locations)

# (3) density
newdensity = density(k3, locations)
```

---

orbital *Data : Normal Vectors to the Orbital Planes of the 9 Planets*

Description

The 9 planets in our solar system are evolving the sun via their own orbits. This data provides normal vector of the orbital planes. Normal vectors are unit-norm vectors, so that they are thought to reside on 2-dimensional sphere.

Usage

```
data(orbital)
```

Format

an (9×3) matrix where each row is a normal vector for a planet.

See Also

wrap.sphere

Examples

```
## LOAD THE DATA AND WRAP AS RIEMOBJ
data(orbital)
myorb = wrap.sphere(orbital)

## VISUALIZE
mds2d = riem.mds(myorb)$embed
opar <- par(no.readonly=TRUE)
plot(mds2d, main="9 Planets", pch=19, xlab="x", ylab="y")
par(opar)
```

---

passiflora *Data : Passiflora Leaves*

Description

Passiflora is a genus of about 550 species of flowering plants. This dataset contains 15 landmarks in 2 dimension of 3319 leaves of 40 species. Papers listed in the reference section analyzed the data and found 7 clusters.

Usage

```
data(passiflora)
```

Format

a named list containing

data a 3d array of size $(15 \times 2 \times 3319)$.

species a length-3319 vector of 40 species factors.

class a length-3319 vector of 7 cluster factors.

References

Chitwood DH, Otoni WC (2017). "Divergent leaf shapes among Passiflora species arise from a shared juvenile morphology." *Plant Direct*, **1**(5), e00028. ISSN 24754455.

Chitwood DH, Otoni WC (2017). "Morphometric analysis of Passiflora leaves: the relationship between landmarks of the vasculature and elliptical Fourier descriptors of the blade." *GigaScience*, **6**(1). ISSN 2047-217X.

See Also

`wrap.landmark`

Examples

```
data(passiflora)                          # load the data
riemobj = wrap.landmark(passiflora$data) # wrap as RIEMOBJ
pga2d   = riem.pga(riemobj)$embed        # embedding via PGA

opar <- par(no.readonly=TRUE)            # visualize
plot(pga2d, col=passiflora$class, pch=19, cex=0.7,
     main="PGA Embedding of Passiflora Leaves",
     xlab="dimension 1", ylab="dimension 2")
par(opar)
```

---

predict.m2skreg *Prediction for Manifold-to-Scalar Kernel Regression*

Description

Given new observations $X_1, X_2, \ldots, X_M \in \mathcal{M}$, plug in the data with respect to the fitted model for prediction.

Usage

```
## S3 method for class 'm2skreg'
predict(object, newdata, geometry = c("intrinsic", "extrinsic"), ...)
```

Arguments

`object`	an object of `m2skreg` class. See `riem.m2skreg` for more details.
`newdata`	a S3 `"riemdata"` class for manifold-valued data corresponding to $X_1, \ldots, X_M$.
`geometry`	(case-insensitive) name of geometry; either geodesic (`"intrinsic"`) or embedded (`"extrinsic"`) geometry.
`...`	further arguments passed to or from other methods.

Value

a length-M vector of predictted values.

See Also

riem.m2skreg

Examples

```
#------------------------------------------------------------------
#                    Example on Sphere S^2
#
#  X : equi-spaced points from (0,0,1) to (0,1,0)
#  y : sin(x) with perturbation
#
#  Our goal is to check whether the predict function works well
#  by comparing the originally predicted values vs. those of the same data.
#------------------------------------------------------------------
# GENERATE DATA
npts = 100
nlev = 0.25
thetas = seq(from=0, to=pi/2, length.out=npts)
Xstack = cbind(rep(0,npts), sin(thetas), cos(thetas))

Xriem  = wrap.sphere(Xstack)
ytrue  = sin(seq(from=0, to=2*pi, length.out=npts))
ynoise = ytrue + rnorm(npts, sd=nlev)

# FIT & PREDICT
obj_fit   = riem.m2skreg(Xriem, ynoise, bandwidth=0.01)
yval_fits = obj_fit$ypred
yval_pred = predict(obj_fit, Xriem)

# VISUALIZE
xgrd <- 1:npts
opar <- par(no.readonly=TRUE)
par(mfrow=c(1,2))
plot(xgrd, yval_fits, pch=19, cex=0.5, "b", xlab="", ylim=c(-2,2), main="original fit")
lines(xgrd, ytrue, col="red", lwd=1.5)
plot(xgrd, yval_pred, pch=19, cex=0.5, "b", xlab="", ylim=c(-2,2), main="from 'predict'")
lines(xgrd, ytrue, col="red", lwd=1.5)
par(opar)
```

---

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.

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

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.coreset18B` *Build Lightweight Coreset*

Description

Given manifold-valued data $X_1, X_2, \ldots, X_N \in \mathcal{M}$, this algorithm finds the coreset of size M that can be considered as a compressed representation according to the lightweight coreset construction scheme proposed by the reference below.

Usage

```
riem.coreset18B(
  riemobj,
  M = length(riemobj$data)/2,
  geometry = c("intrinsic", "extrinsic"),
  ...
)
```

Arguments

`riemobj`	a S3 `"riemdata"` class for N manifold-valued data.
`M`	the size of coreset (default: $N/2$).
`geometry`	(case-insensitive) name of geometry; either geodesic (`"intrinsic"`) or embedded (`"extrinsic"`) geometry.
`...`	extra parameters including **maxiter** maximum number of iterations to be run (default:50). **eps** tolerance level for stopping criterion (default: 1e-5).

Value

a named list containing

coreid a length-M index vector of the coreset.

weight a length-M vector of weights for each element.

References

Bachem O, Lucic M, Krause A (2018). "Scalable k -Means Clustering via Lightweight Coresets." In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 1119–1127. ISBN 978-1-4503-5552-0.

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)

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

```
## FIND CORESET OF SIZES 3, 6, 9
core1 = riem.coreset18B(myriem, M=3)
core2 = riem.coreset18B(myriem, M=6)
core3 = riem.coreset18B(myriem, M=9)

col1 = rep(1,30); col1[core1$coreid] = 2
col2 = rep(1,30); col2[core2$coreid] = 2
col3 = rep(1,30); col3[core3$coreid] = 2

## VISUALIZE
opar <- par(no.readonly=TRUE)
par(mfrow=c(1,3), pty="s")
plot(embed2, pch=19, col=col1, main="coreset size=3")
plot(embed2, pch=19, col=col2, main="coreset size=6")
plot(embed2, pch=19, col=col3, main="coreset size=9")
par(opar)
```

`riem.distlp` *Distance between Two Curves on Manifolds*

Description

Given two curves $\gamma_1, \gamma_2 : I \to \mathcal{M}$, we are interested in measuring the discrepancy of two curves. Usually, data are given as discrete observations so we are offering several methods to perform the task. See the section below for detailed description.

Usage

```
riem.distlp(
  riemobj1,
  riemobj2,
  vect = NULL,
  geometry = c("intrinsic", "extrinsic"),
  ...
)
```

Arguments

`riemobj1`	a S3 `"riemdata"` class for N manifold-valued data along the curve.
`riemobj2`	a S3 `"riemdata"` class for N manifold-valued data along the curve.
`vect`	a vector of domain values. If given `Null` (default), sequence `1:N` is set.
`geometry`	(case-insensitive) name of geometry; either geodesic (`"intrinsic"`) or embedded (`"extrinsic"`) geometry.
`...`	extra parameters including **p** an exponent (default: 2).

Value

the distance value.

Default Method

Trapezoidal Approximation Assume $\gamma_1(t_i) = X_i$ and $\gamma_2(t_i) = Y_i$ for $i = 1, 2, \ldots, N$. In the Euclidean space, L_p distance between two scalar-valued functions is defined as

$$L_p^p(\gamma_1(x), \gamma_2(x) = \int_{\mathcal{X}} |\gamma_1(x) - \gamma_2(x)|^p dx$$

. We extend this approach to manifold-valued curves

$$L_p^p(\gamma_1(t), \gamma_2(t)) = \int_{t \in I} d^p(\gamma_1(t), \gamma_2(t)) dt$$

where $d(\cdot, \cdot)$ is an intrinsic/extrinsic distance on manifolds. With the given representations, the above integral is approximated using trapezoidal rule.

Examples

```
#-----------------------------------------------------------------
#                          Curves on Sphere
#
#  curve1 : y = 0.5*cos(x) on the tangent space at (0,0,1)
#  curve2 : y = 0.5*cos(x) on the tangent space at (0,0,1)
#  curve3 : y = 0.5*sin(x) on the tangent space at (0,0,1)
#
# * distance between curve1 & curve2 should be close to 0.
# * distance between curve1 & curve3 should be large.
#-----------------------------------------------------------------
## GENERATION
vecx  = seq(from=-0.9, to=0.9, length.out=50)
vecy1 = 0.5*cos(vecx) + rnorm(50, sd=0.05)
vecy2 = 0.5*cos(vecx) + rnorm(50, sd=0.05)
vecy3 = 0.5*sin(vecx) + rnorm(50, sd=0.05)

## WRAP AS RIEMOBJ
mat1 = cbind(vecx, vecy1, 1); mat1 = mat1/sqrt(rowSums(mat1^2))
mat2 = cbind(vecx, vecy2, 1); mat2 = mat2/sqrt(rowSums(mat2^2))
mat3 = cbind(vecx, vecy3, 1); mat3 = mat3/sqrt(rowSums(mat3^2))

rcurve1 = wrap.sphere(mat1)
rcurve2 = wrap.sphere(mat2)
rcurve3 = wrap.sphere(mat3)

## COMPUTE DISTANCES
riem.distlp(rcurve1, rcurve2, vect=vecx)
riem.distlp(rcurve1, rcurve3, vect=vecx)
```

`riem.dtw` *Dynamic Time Warping Distance*

Description

Given two time series - a query $X = (X_1, X_2, \ldots, X_N)$ and a reference $Y = (Y_1, Y_2, \ldots, Y_M)$, `riem.dtw` computes the most basic version of Dynamic Time Warping (DTW) distance between two series using a symmetric step pattern, meaning no window constraints and others at all. Although the scope of DTW in Euclidean space-valued objects is rich, it is scarce for manifold-valued curves. If you are interested in the topic, we refer to **dtw** package.

Usage

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

Arguments

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

Value

the distance value.

Examples

```
#-------------------------------------------------------------------
#                          Curves on Sphere
#
#  curve1 : y = 0.5*cos(x) on the tangent space at (0,0,1)
#  curve2 : y = 0.5*sin(x) on the tangent space at (0,0,1)
#
#  we will generate two sets for curves of different sizes.
#-------------------------------------------------------------------
## GENERATION
clist = list()
for (i in 1:10){ # curve type 1
  vecx = seq(from=-0.9, to=0.9, length.out=sample(10:50, 1))
  vecy = 0.5*cos(vecx) + rnorm(length(vecx), sd=0.1)
  mats = cbind(vecx, vecy, 1)
  clist[[i]] = wrap.sphere(mats/sqrt(rowSums(mats^2)))
}
for (i in 1:10){ # curve type 2
  vecx = seq(from=-0.9, to=0.9, length.out=sample(10:50, 1))
  vecy = 0.5*sin(vecx) + rnorm(length(vecx), sd=0.1)
  mats = cbind(vecx, vecy, 1)
```

```
  clist[[i+10]] = wrap.sphere(mats/sqrt(rowSums(mats^2)))
}

## COMPUTE DISTANCES
outint = array(0,c(20,20))
outext = array(0,c(20,20))
for (i in 1:19){
  for (j in 2:20){
    outint[i,j] <- outint[j,i] <- riem.dtw(clist[[i]], clist[[j]],
                                           geometry="intrinsic")
    outext[i,j] <- outext[j,i] <- riem.dtw(clist[[i]], clist[[j]],
                                           geometry="extrinsic")
  }
}

## VISUALIZE
opar <- par(no.readonly=TRUE)
par(mfrow=c(1,2), pty="s")
image(outint[,20:1], axes=FALSE, main="intrinsic DTW Distance")
image(outext[,20:1], axes=FALSE, main="extrinsic DTW Distance")
par(opar)
```

`riem.fanova` *Fréchet Analysis of Variance*

Description

Given sets of manifold-valued data $X_{1:n_1}^{(1)}, X_{1:n_2}^{(2)}, \ldots, X_{1:n_m}^{(m)}$, 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.

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  = 200
class1 = list()
class2 = list()
class3 = list()
class4 = list()
for (i in 1:ndata){
  tmpxy = matrix(rnorm(4*2, sd=0.1), ncol=2)
  tmpz  = rep(1,4)
  tmp3d = cbind(tmpxy, tmpz)
  tmp  = tmp3d/sqrt(rowSums(tmp3d^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=999)
```

`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.

References

Müllner D (2013). "fastcluster : Fast Hierarchical, Agglomerative Clustering Routines for R and Python." *Journal of Statistical Software*, **53**(9). ISSN 1548-7660.

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.isomap` *Isometric Feature Mapping*

Description

ISOMAP - isometric feature mapping - is a dimensionality reduction method to apply classical multidimensional scaling to the geodesic distance that is computed on a weighted nearest neighborhood graph. Nearest neighbor is defined by k-NN where two observations are said to be connected when they are mutually included in each other's nearest neighbor. Note that it is possible for geodesic distances to be `Inf` when nearest neighbor graph construction incurs separate connected components. When an extra parameter `padding=TRUE`, infinite distances are replaced by 2 times the maximal finite geodesic distance.

Usage

```
riem.isomap(
  riemobj,
  ndim = 2,
  nnbd = 5,
  geometry = c("intrinsic", "extrinsic"),
  ...
)
```

Arguments

`riemobj`	a S3 `"riemdata"` class for N manifold-valued data.
`ndim`	an integer-valued target dimension (default: 2).
`nnbd`	the size of nearest neighborhood (default: 5).
`geometry`	(case-insensitive) name of geometry; either geodesic (`"intrinsic"`) or embedded (`"extrinsic"`) geometry.
`...`	extra parameters including
	padding a logical; if `TRUE`, `Inf`-valued geodesic distances are replaced by 2 times the maximal geodesic distance in the data.

Value

a named list containing

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

References

Silva VD, Tenenbaum JB (2003). "Global Versus Local Methods in Nonlinear Dimensionality Reduction." In Becker S, Thrun S, Obermayer K (eds.), *Advances in Neural Information Processing Systems 15*, 721–728. MIT Press.

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 AND ISOMAP WITH DIFFERENT NEIGHBORHOOD SIZE
mdss = riem.mds(myriem)$embed
iso1 = riem.isomap(myriem, nnbd=5)$embed
iso2 = riem.isomap(myriem, nnbd=10)$embed

## VISUALIZE
opar = par(no.readonly=TRUE)
par(mfrow=c(1,3), pty="s")
plot(mdss, col=mylabs, pch=19, main="MDS")
plot(iso1, col=mylabs, pch=19, main="ISOMAP:nnbd=5")
plot(iso2, col=mylabs, pch=19, main="ISOMAP:nnbd=10")
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"), ...)
```

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.
`...`	extra parameters including
	algorithm (case-insensitive) name of an algorithm; `"MacQueen"` (default), or `"Lloyd"`.
	init (case-insensitive) name of an initialization scheme; `"plus"` for k-means++ (default), or `"random"`.
	maxiter maximum number of iterations to be run (default:50).
	nstart the number of random starts (default: 5).

Value

a named list containing

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

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

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.

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-MEANS 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.kmeans18B *K-Means Clustering with Lightweight Coreset*

Description

The modified version of lightweight coreset for scalable k-means computation is applied for manifold-valued data $X_1, X_2, \ldots, X_N \in \mathcal{M}$. The smaller the set is, the faster the execution becomes with potentially larger quantization errors.

Usage

```
riem.kmeans18B(
  riemobj,
  k = 2,
  M = length(riemobj$data)/2,
  geometry = c("intrinsic", "extrinsic"),
  ...
)
```

Arguments

`riemobj`	a S3 `"riemdata"` class for N manifold-valued data.
`k`	the number of clusters.
`M`	the size of coreset (default: $N/2$).
`geometry`	(case-insensitive) name of geometry; either geodesic (`"intrinsic"`) or embedded (`"extrinsic"`) geometry.
`...`	extra parameters including **maxiter** maximum number of iterations to be run (default:50). **nstart** the number of random starts (default: 5).

Value

a named list containing

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

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

score within-cluster sum of squares (WCSS).

References

Bachem O, Lucic M, Krause A (2018). "Scalable k -Means Clustering via Lightweight Coresets." In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 1119–1127. ISBN 978-1-4503-5552-0.

See Also

`riem.coreset18B`

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)

## TRY DIFFERENT SIZES OF CORESET WITH K=4 FIXED
core1 = riem.kmeans18B(myriem, k=3, M=5)
core2 = riem.kmeans18B(myriem, k=3, M=10)
core3 = riem.kmeans18B(myriem, k=3, M=15)

## 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="kmeans18B: M=5",  col=core1$cluster)
plot(mds2d, pch=19, main="kmeans18B: M=10", col=core2$cluster)
plot(mds2d, pch=19, main="kmeans18B: M=15", col=core3$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.

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.

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.

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.kpca` *Kernel Principal Component Analysis*

Description

Although the method of Kernel Principal Component Analysis (KPCA) was originally developed to visualize non-linearly distributed data in Euclidean space, we graft this to the case for manifolds where extrinsic geometry is explicitly available. The algorithm uses Gaussian kernel with

$$K(X_i, X_j) = \exp\left(-\frac{d^2(X_i, X_j)}{2\sigma^2}\right)$$

where σ is a bandwidth parameter and $d(\cdot,\cdot)$ is an extrinsic distance defined on a specific manifold.

Usage

```
riem.kpca(riemobj, ndim = 2, sigma = 1)
```

Arguments

`riemobj`	a S3 `"riemdata"` class for N manifold-valued data.
`ndim`	an integer-valued target dimension (default: 2).
`sigma`	the bandwidth parameter (default: 1).

Value

a named list containing

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

vars a length-N vector of eigenvalues from kernelized covariance matrix.

References

Schölkopf B, Smola A, Müller K (1997). "Kernel principal component analysis." In Goos G, Hartmanis J, van Leeuwen J, Gerstner W, Germond A, Hasler M, Nicoud J (eds.), *Artificial Neural Networks — ICANN'97*, volume 1327, 583–588. Springer Berlin Heidelberg, Berlin, Heidelberg. ISBN 978-3-540-63631-1 978-3-540-69620-9.

Examples

```
#------------------------------------------------------------------
#          Example for Gorilla Skull Data : 'gorilla'
#------------------------------------------------------------------
## PREPARE THE DATA
#  Aggregate two classes into one set
data(gorilla)

mygorilla = array(0,c(8,2,59))
for (i in 1:29){
  mygorilla[,,i] = gorilla$male[,,i]
}
for (i in 30:59){
  mygorilla[,,i] = gorilla$female[,,i-29]
}

gor.riem = wrap.landmark(mygorilla)
gor.labs = c(rep("red",29), rep("blue",30))

## APPLY KPCA WITH DIFFERENT KERNEL BANDWIDTHS
kpca1 = riem.kpca(gor.riem, sigma=0.01)
kpca2 = riem.kpca(gor.riem, sigma=1)
kpca3 = riem.kpca(gor.riem, sigma=100)
## VISUALIZE
opar <- par(no.readonly=TRUE)
par(mfrow=c(1,3), pty="s")
plot(kpca1$embed, pch=19, col=gor.labs, main="sigma=1/100")
plot(kpca2$embed, pch=19, col=gor.labs, main="sigma=1")
plot(kpca3$embed, pch=19, col=gor.labs, main="sigma=100")
par(opar)
```

`riem.m2skreg` *Manifold-to-Scalar Kernel Regression*

Description

Given N observations $X_1, X_2, \ldots, X_N \in \mathcal{M}$ and scalars $y_1, y_2, \ldots, y_N \in \mathbf{R}$, perform the Nadaraya-Watson kernel regression by

$$\hat{m}_h(X) = \frac{\sum_{i=1}^{n} K\left(\frac{d(X,X_i)}{h}\right) y_i}{\sum_{i=1}^{n} K\left(\frac{d(X,X_i)}{h}\right)}$$

where the Gaussian kernel is defined as

$$K(x) := \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right)$$

with the bandwidth parameter $h > 0$ that controls the degree of smoothness.

Usage

```
riem.m2skreg(
  riemobj,
  y,
  bandwidth = 0.5,
  geometry = c("intrinsic", "extrinsic")
)
```

Arguments

`riemobj`	a S3 `"riemdata"` class for N manifold-valued data corresponding to $X_1, \ldots, X_N$.
`y`	a length-N vector of dependent variable values.
`bandwidth`	a nonnegative number that controls smoothness.
`geometry`	(case-insensitive) name of geometry; either geodesic (`"intrinsic"`) or embedded (`"extrinsic"`) geometry.

Value

a named list of S3 class `m2skreg` containing

ypred a length-N vector of smoothed responses.

bandwidth the bandwidth value that was originally provided, which is saved for future use.

inputs a list containing both `riemobj` and `y` for future use.

Examples

```
#-----------------------------------------------------------------------
#                    Example on Sphere S^2
#
#  X : equi-spaced points from (0,0,1) to (0,1,0)
#  y : sin(x) with perturbation
#-----------------------------------------------------------------------
# GENERATE DATA
npts = 100
nlev = 0.25
thetas = seq(from=0, to=pi/2, length.out=npts)
Xstack = cbind(rep(0,npts), sin(thetas), cos(thetas))

Xriem  = wrap.sphere(Xstack)
ytrue  = sin(seq(from=0, to=2*pi, length.out=npts))
ynoise = ytrue + rnorm(npts, sd=nlev)
```

```
# FIT WITH DIFFERENT BANDWIDTHS
fit1 = riem.m2skreg(Xriem, ynoise, bandwidth=0.001)
fit2 = riem.m2skreg(Xriem, ynoise, bandwidth=0.01)
fit3 = riem.m2skreg(Xriem, ynoise, bandwidth=0.1)

# VISUALIZE
xgrd <- 1:npts
opar <- par(no.readonly=TRUE)
par(mfrow=c(1,3))
plot(xgrd, fit1$ypred, pch=19, cex=0.5, "b", xlab="", ylim=c(-2,2), main="h=1e-3")
lines(xgrd, ytrue, col="red", lwd=1.5)
plot(xgrd, fit2$ypred, pch=19, cex=0.5, "b", xlab="", ylim=c(-2,2), main="h=1e-2")
lines(xgrd, ytrue, col="red", lwd=1.5)
plot(xgrd, fit3$ypred, pch=19, cex=0.5, "b", xlab="", ylim=c(-2,2), main="h=1e-1")
lines(xgrd, ytrue, col="red", lwd=1.5)
par(opar)
```

`riem.m2skregCV` *Manifold-to-Scalar Kernel Regression with K-Fold Cross Validation*

Description

Manifold-to-Scalar Kernel Regression with K-Fold Cross Validation

Usage

```
riem.m2skregCV(
  riemobj,
  y,
  bandwidths = seq(from = 0.01, to = 1, length.out = 10),
  geometry = c("intrinsic", "extrinsic"),
  kfold = 5
)
```

Arguments

`riemobj`	a S3 `"riemdata"` class for N manifold-valued data corresponding to $X_1, \ldots, X_N$.
`y`	a length-N vector of dependent variable values.
`bandwidths`	a vector of nonnegative numbers that control smoothness.
`geometry`	(case-insensitive) name of geometry; either geodesic (`"intrinsic"`) or embedded (`"extrinsic"`) geometry.
`kfold`	the number of folds for cross validation.

Value

a named list of S3 class `m2skreg` containing

ypred a length-N vector of optimal smoothed responses.

bandwidth the optimal bandwidth value.

inputs a list containing both `riemobj` and `y` for future use.

errors a matrix whose columns are `bandwidths` values and corresponding errors measure in SSE.

Examples

```
#-----------------------------------------------------------------
#                    Example on Sphere S^2
#
#  X : equi-spaced points from (0,0,1) to (0,1,0)
#  y : sin(x) with perturbation
#-----------------------------------------------------------------
# GENERATE DATA
set.seed(496)
npts = 100
nlev = 0.25
thetas = seq(from=0, to=pi/2, length.out=npts)
Xstack = cbind(rep(0,npts), sin(thetas), cos(thetas))

Xriem  = wrap.sphere(Xstack)
ytrue  = sin(seq(from=0, to=2*pi, length.out=npts))
ynoise = ytrue + rnorm(npts, sd=nlev)

# FIT WITH 5-FOLD CV
cv_band = (10^seq(from=-4, to=-1, length.out=200))
cv_fit  = riem.m2skregCV(Xriem, ynoise, bandwidths=cv_band)
cv_err  = cv_fit$errors

# VISUALIZE
opar <- par(no.readonly=TRUE)
par(mfrow=c(1,2))
plot(1:npts, cv_fit$ypred, pch=19, cex=0.5, "b", xlab="", main="optimal prediction")
lines(1:npts, ytrue, col="red", lwd=1.5)
plot(cv_err[,1], cv_err[,2], "b", pch=19, cex=0.5, main="5-fold CV errors",
     xlab="bandwidth", ylab="SSE")
abline(v=cv_fit$bandwidth, col="blue", lwd=1.5)
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 (default: 2).
`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.

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 ($w_i = 1/n$) for all observations.

Usage

```
riem.mean(riemobj, weight = NULL, geometry = c("intrinsic", "extrinsic"), ...)
```

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.
`...`	extra parameters including **maxiter** maximum number of iterations to be run (default:50). **eps** tolerance level for stopping criterion (default: 1e-5).

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"),
  ...
)
```

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.
...	extra parameters including **maxiter** maximum number of iterations to be run (default:50). **eps** tolerance level for stopping criterion (default: 1e-5).

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.

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 PT, Lu C, Pizer SM, 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.

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.phate` *PHATE*

Description

PHATE is a nonlinear manifold learning method that is specifically targeted at improving diffusion maps by incorporating data-adaptive kernel construction, detection of optimal time scale, and information-theoretic metric measures.

Usage

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

Arguments

`riemobj`	a S3 `"riemdata"` class for N manifold-valued data.
`ndim`	an integer-valued target dimension (default: 2).
`geometry`	(case-insensitive) name of geometry; either geodesic (`"intrinsic"`) or embedded (`"extrinsic"`) geometry.
`...`	extra parameters for `PHATE` including
	nbdk size of nearest neighborhood (default: 5).
	alpha decay parameter for Gaussian kernel exponent (default: 2).
	potential type of potential distance transformation; `"log"` or `"sqrt"` (default: `"log"`).

Value

a named list containing

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

References

Moon KR, van Dijk D, Wang Z, Gigante S, Burkhardt DB, Chen WS, Yim K, van den Elzen A, Hirn MJ, Coifman RR, Ivanova NB, Wolf G, Krishnaswamy S (2019). "Visualizing Structure and Transitions in High-Dimensional Biological Data." *Nature Biotechnology*, **37**(12), 1482–1492. ISSN 1087-0156, 1546-1696.

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)
```

```
## PHATE EMBEDDING WITH LOG & SQRT POTENTIAL
phate_log  = riem.phate(myriem, potential="log")$embed
phate_sqrt = riem.phate(myriem, potential="sqrt")$embed
embed_mds  = riem.mds(myriem)$embed

## VISUALIZE
opar = par(no.readonly=TRUE)
par(mfrow=c(1,3), pty="s")
plot(embed_mds,  col=mylabs, pch=19, main="MDS" )
plot(phate_log,  col=mylabs, pch=19, main="PHATE+Log")
plot(phate_sqrt, col=mylabs, pch=19, main="PHATE+Sqrt")
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.

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.sammon` *Sammon Mapping*

Description

Given N observations $X_1, X_2, \ldots, X_N \in \mathcal{M}$, apply Sammon mapping, a non-linear dimensionality reduction method. Since the method depends only on the pairwise distances of the data, it can be adapted to the manifold-valued data.

Usage

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

Arguments

`riemobj`	a S3 `"riemdata"` class for N manifold-valued data.
`ndim`	an integer-valued target dimension (default: 2).
`geometry`	(case-insensitive) name of geometry; either geodesic (`"intrinsic"`) or embedded (`"extrinsic"`) geometry.
`...`	extra parameters including
	maxiter maximum number of iterations to be run (default:50).
	eps tolerance level for stopping criterion (default: 1e-5).

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

Sammon JW (1969). "A Nonlinear Mapping for Data Structure Analysis." *IEEE Transactions on Computers*, **C-18**(5), 401–409. ISSN 0018-9340.

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)

## COMPARE SAMMON WITH MDS
embed2mds = riem.mds(myriem, ndim=2)$embed
embed2sam = riem.sammon(myriem, ndim=2)$embed
```

```
## VISUALIZE
opar = par(no.readonly=TRUE)
par(mfrow=c(1,2), pty="s")
plot(embed2mds, col=mylabs, pch=19, main="MDS")
plot(embed2sam, col=mylabs, pch=19, main="Sammon mapping")
par(opar)
```

`riem.sc05Z` *Spectral Clustering by Zelnik-Manor and Perona (2005)*

Description

Zelnik-Manor and Perona proposed a method to define a set of data-driven bandwidth parameters where σ_i is the distance from a point x_i to its nnbd-th nearest neighbor. Then the affinity matrix is defined as

$$A_{ij} = \exp(-d(x_i, d_j)^2/\sigma_i\sigma_j)$$

and the standard spectral clustering of Ng, Jordan, and Weiss (`riem.scNJW`) is applied.

Usage

```
riem.sc05Z(riemobj, k = 2, nnbd = 7, geometry = c("intrinsic", "extrinsic"))
```

Arguments

`riemobj`	a S3 `"riemdata"` class for N manifold-valued data.
`k`	the number of clusters (default: 2).
`nnbd`	neighborhood size to define data-driven bandwidth parameter (default: 7).
`geometry`	(case-insensitive) name of geometry; either geodesic (`"intrinsic"`) or embedded (`"extrinsic"`) geometry.

Value

a named list containing

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

eigval eigenvalues of the graph laplacian's spectral decomposition.

embeds an $(N \times k)$ low-dimensional embedding.

References

Zelnik-manor L, Perona P (2005). "Self-Tuning Spectral Clustering." In Saul LK, Weiss Y, Bottou L (eds.), *Advances in Neural Information Processing Systems 17*, 1601–1608. MIT Press.

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)
lab    = rep(c(1,2,3), each=10)

## CLUSTERING WITH DIFFERENT K VALUES
cl2 = riem.sc05Z(myriem, k=2)$cluster
cl3 = riem.sc05Z(myriem, k=3)$cluster
cl4 = riem.sc05Z(myriem, k=4)$cluster

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

## VISUALIZE
opar <- par(no.readonly=TRUE)
par(mfrow=c(1,4), pty="s")
plot(mds2d, col=lab, pch=19, main="true label")
plot(mds2d, col=cl2, pch=19, main="riem.sc05Z: k=2")
plot(mds2d, col=cl3, pch=19, main="riem.sc05Z: k=3")
plot(mds2d, col=cl4, pch=19, main="riem.sc05Z: k=4")
par(opar)
```

`riem.scNJW` *Spectral Clustering by Ng, Jordan, and Weiss (2002)*

Description

The version of Ng, Jordan, and Weiss first constructs the affinity matrix

$$A_{ij} = \exp(-d(x_i, d_j)^2/\sigma^2)$$

where σ is a common bandwidth parameter and performs k-means clustering on the row-space of eigenvectors for the symmetric graph laplacian matrix

$$L = D^{-1/2}(D - A)D^{-1/2}$$

.

Usage

```
riem.scNJW(riemobj, k = 2, sigma = 1, geometry = c("intrinsic", "extrinsic"))
```

Arguments

`riemobj`	a S3 `"riemdata"` class for N manifold-valued data.
`k`	the number of clusters (default: 2).
`sigma`	bandwidth parameter (default: 1).
`geometry`	(case-insensitive) name of geometry; either geodesic (`"intrinsic"`) or embedded (`"extrinsic"`) geometry.

Value

a named list containing

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

eigval eigenvalues of the graph laplacian's spectral decomposition.

embeds an $(N \times k)$ low-dimensional embedding.

References

Ng AY, Jordan MI, Weiss Y (2002). "On Spectral Clustering: Analysis and an Algorithm." In Dietterich TG, Becker S, Ghahramani Z (eds.), *Advances in Neural Information Processing Systems 14*, 849–856. MIT Press.

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)
lab    = rep(c(1,2,3), each=10)

## CLUSTERING WITH DIFFERENT K VALUES
cl2 = riem.scNJW(myriem, k=2)$cluster
cl3 = riem.scNJW(myriem, k=3)$cluster
cl4 = riem.scNJW(myriem, k=4)$cluster

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

## VISUALIZE
opar <- par(no.readonly=TRUE)
par(mfrow=c(1,4), pty="s")
plot(mds2d, col=lab, pch=19, main="true label")
plot(mds2d, col=cl2, pch=19, main="riem.scNJW: k=2")
plot(mds2d, col=cl3, pch=19, main="riem.scNJW: k=3")
plot(mds2d, col=cl4, pch=19, main="riem.scNJW: k=4")
par(opar)
```

`riem.scSM` *Spectral Clustering by Shi and Malik (2000)*

Description

The version of Shi and Malik first constructs the affinity matrix

$$A_{ij} = \exp(-d(x_i, d_j)^2/\sigma^2)$$

where σ is a common bandwidth parameter and performs k-means clustering on the row-space of eigenvectors for the random-walk graph laplacian matrix

$$L = D^{-1}(D - A)$$

.

Usage

```
riem.scSM(riemobj, k = 2, sigma = 1, geometry = c("intrinsic", "extrinsic"))
```

Arguments

`riemobj`	a S3 `"riemdata"` class for N manifold-valued data.
`k`	the number of clusters (default: 2).
`sigma`	bandwidth parameter (default: 1).
`geometry`	(case-insensitive) name of geometry; either geodesic (`"intrinsic"`) or embedded (`"extrinsic"`) geometry.

Value

a named list containing

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

eigval eigenvalues of the graph laplacian's spectral decomposition.

embeds an $(N \times k)$ low-dimensional embedding.

References

Shi J, Malik J (2000). "Normalized Cuts and Image Segmentation." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905.

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)
lab    = rep(c(1,2,3), each=10)

## CLUSTERING WITH DIFFERENT K VALUES
cl2 = riem.scSM(myriem, k=2)$cluster
cl3 = riem.scSM(myriem, k=3)$cluster
cl4 = riem.scSM(myriem, k=4)$cluster
```

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

## VISUALIZE
opar <- par(no.readonly=TRUE)
par(mfrow=c(1,4), pty="s")
plot(mds2d, col=lab, pch=19, main="true label")
plot(mds2d, col=cl2, pch=19, main="riem.scSM: k=2")
plot(mds2d, col=cl3, pch=19, main="riem.scSM: k=3")
plot(mds2d, col=cl4, pch=19, main="riem.scSM: k=4")
par(opar)
```

`riem.scUL` *Spectral Clustering with Unnormalized Laplacian*

Description

The version of Shi and Malik first constructs the affinity matrix

$$A_{ij} = \exp(-d(x_i, d_j)^2/\sigma^2)$$

where σ is a common bandwidth parameter and performs k-means clustering on the row-space of eigenvectors for the unnormalized graph laplacian matrix

$$L = D - A$$

.

Usage

```
riem.scUL(riemobj, k = 2, sigma = 1, geometry = c("intrinsic", "extrinsic"))
```

Arguments

`riemobj`	a S3 `"riemdata"` class for N manifold-valued data.
`k`	the number of clusters (default: 2).
`sigma`	bandwidth parameter (default: 1).
`geometry`	(case-insensitive) name of geometry; either geodesic (`"intrinsic"`) or embedded (`"extrinsic"`) geometry.

Value

a named list containing

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

eigval eigenvalues of the graph laplacian's spectral decomposition.

embeds an $(N \times k)$ low-dimensional embedding.

References

von Luxburg U (2007). "A Tutorial on Spectral Clustering." *Statistics and Computing*, 17(4):395–416.

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)
lab    = rep(c(1,2,3), each=10)

## CLUSTERING WITH DIFFERENT K VALUES
cl2 = riem.scUL(myriem, k=2)$cluster
cl3 = riem.scUL(myriem, k=3)$cluster
cl4 = riem.scUL(myriem, k=4)$cluster

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

## VISUALIZE
opar <- par(no.readonly=TRUE)
par(mfrow=c(1,4), pty="s")
plot(mds2d, col=lab, pch=19, main="true label")
plot(mds2d, col=cl2, pch=19, main="riem.scUL: k=2")
plot(mds2d, col=cl3, pch=19, main="riem.scUL: k=3")
plot(mds2d, col=cl4, pch=19, main="riem.scUL: k=4")
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"), ...)
```

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).
`...`	extra parameters including
	maxiter maximum number of iterations to be run (default:50).
	eps tolerance level for stopping criterion (default: 1e-5).

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.

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.test2bg14` *Two-Sample Test modified from Biswas and Ghosh (2014)*

Description

Given M observations $X_1, X_2, \ldots, X_M \in \mathcal{M}$ and N observations $Y_1, Y_2, \ldots, Y_N \in \mathcal{M}$, perform the permutation test of equal distribution

$$H_0 \ : \ \mathcal{P}_X = \mathcal{P}_Y$$

by the method from Biswas and Ghosh (2014). The method, originally proposed for Euclidean-valued data, is adapted to the general Riemannian manifold with intrinsic/extrinsic distance.

Usage

```
riem.test2bg14(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.
`...`	extra parameters including
	nperm the number of permutations (default: 999).

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

Biswas M, Ghosh AK (2014). "A nonparametric two-sample test applicable to high dimensional data." *Journal of Multivariate Analysis*, **123**, 160–171. ISSN 0047259X.

You K, Park H (2020). "Re-visiting Riemannian geometry of symmetric positive definite matrices for the analysis of functional connectivity." *NeuroImage*, 117464. ISSN 10538119.

Examples

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

## PERFORM PERMUTATION TEST
#  it is expected to return a very small number.

riem.test2bg14(myriem1, myriem2, nperm=999)


## Not run:
## CHECK WITH EMPIRICAL TYPE-1 ERROR
set.seed(777)
ntest = 1000
pvals = rep(0,ntest)
```

```
for (i in 1:ntest){
  X = cbind(matrix(rnorm(30*2, sd=0.1),ncol=2), rep(1,30))
  Y = cbind(matrix(rnorm(30*2, sd=0.1),ncol=2), rep(1,30))
  Xnorm = X/sqrt(rowSums(X^2))
  Ynorm = Y/sqrt(rowSums(Y^2))

  Xriem = wrap.sphere(Xnorm)
  Yriem = wrap.sphere(Ynorm)
  pvals[i] = riem.test2bg14(Xriem, Yriem, nperm=999)$p.value
}

emperr = round(sum((pvals <= 0.05))/ntest, 5)
print(paste0("* EMPIRICAL TYPE-1 ERROR=", emperr))

## End(Not run)
```

`riem.test2wass` *Two-Sample Test with Wasserstein Metric*

Description

Given M observations $X_1, X_2, \ldots, X_M \in \mathcal{M}$ and N observations $Y_1, Y_2, \ldots, Y_N \in \mathcal{M}$, permutation test based on the Wasserstein metric (see `riem.wasserstein` for more details) is applied to test whether two distributions are same or not, i.e.,

$$H_0 \ : \ \mathcal{P}_X = \mathcal{P}_Y$$

with Wasserstein metric $\mathcal{W}_p$ being the measure of discrepancy between two samples.

Usage

```
riem.test2wass(
  riemobj1,
  riemobj2,
  p = 2,
  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.
`p`	an exponent for Wasserstein distance $\mathcal{W}_p$ (default: 2).
`geometry`	(case-insensitive) name of geometry; either geodesic (`"intrinsic"`) or embedded (`"extrinsic"`) geometry.
`...`	extra parameters including

nperm the number of permutations (default: 999).

use.smooth a logical; TRUE to use a smoothed Wasserstein distance, FALSE otherwise.

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.

Examples

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

## PERFORM PERMUTATION TEST
#  it is expected to return a very small number, but
#  small number of 'nperm' may not give a reasonable p-value.

riem.test2wass(myriem1, myriem2, nperm=99, use.smooth=FALSE)


## Not run:
## CHECK WITH EMPIRICAL TYPE-1 ERROR
set.seed(777)
ntest = 1000
pvals = rep(0,ntest)

for (i in 1:ntest){
  X = cbind(matrix(rnorm(30*2, sd=0.1),ncol=2), rep(1,30))
```

```
  Y = cbind(matrix(rnorm(30*2, sd=0.1),ncol=2), rep(1,30))
  Xnorm = X/sqrt(rowSums(X^2))
  Ynorm = Y/sqrt(rowSums(Y^2))

  Xriem = wrap.sphere(Xnorm)
  Yriem = wrap.sphere(Ynorm)
  pvals[i] = riem.test2wass(Xriem, Yriem, nperm=999)$p.value
  print(paste0("iteration ",i,"/",ntest," complete.."))
}

emperr = round(sum((pvals <= 0.05))/ntest, 5)
print(paste0("* EMPIRICAL TYPE-1 ERROR=", emperr))

## End(Not run)
```

`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 from **Rtsne** package, 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.

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)
```

`riem.wasserstein` *Wasserstein Distance between Empirical Measures*

Description

Given two empirical measures μ, ν consisting of M and N observations, p-Wasserstein distance for $p \geq 1$ between two empirical measures is defined as

$$\mathcal{W}_p(\mu, \nu) = \left(\inf_{\gamma \in \Gamma(\mu,\nu)} \int_{\mathcal{M}\times\mathcal{M}} d(x,y)^p d\gamma(x,y) \right)^{1/p}$$

where $\Gamma(\mu, \nu)$ denotes the collection of all measures/couplings on $\mathcal{M} \times \mathcal{M}$ whose marginals are μ and ν on the first and second factors, respectively.

Usage

```
riem.wasserstein(
  riemobj1,
  riemobj2,
  p = 2,
  geometry = c("intrinsic", "extrinsic"),
  ...
)
```

Arguments

`riemobj1`	a S3 `"riemdata"` class for M manifold-valued data, which are atoms of μ.
`riemobj2`	a S3 `"riemdata"` class for N manifold-valued data, which are atoms of ν.
`p`	an exponent for Wasserstein distance $\mathcal{W}_p$ (default: 2).
`geometry`	(case-insensitive) name of geometry; either geodesic (`"intrinsic"`) or embedded (`"extrinsic"`) geometry.
`...`	extra parameters including
	weight1 a length-M weight vector for μ; if `NULL` (default), uniform weight is set.
	weight2 a length-N weight vector for ν; if `NULL` (default), uniform weight is set.

Value

a named list containing

distance $\mathcal{W}_{\surd}$ distance between two empirical measures.

plan an $(M \times N)$ matrix whose rowSums and columnSums are `weight1` and `weight2` respectively.

Examples

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

```
myriem1 = wrap.sphere(mydata1)
myriem2 = wrap.sphere(mydata2)

## COMPUTE p-WASSERSTEIN DISTANCES
dist1 = riem.wasserstein(myriem1, myriem2, p=1)
dist2 = riem.wasserstein(myriem1, myriem2, p=2)
dist5 = riem.wasserstein(myriem1, myriem2, p=5)

pm1 = paste0("p=1: dist=",round(dist1$distance,3))
pm2 = paste0("p=2: dist=",round(dist2$distance,3))
pm5 = paste0("p=5: dist=",round(dist5$distance,3))

## VISUALIZE TRANSPORT PLAN AND DISTANCE
opar <- par(no.readonly=TRUE)
par(mfrow=c(1,3))
image(dist1$plan, axes=FALSE, main=pm1)
image(dist2$plan, axes=FALSE, main=pm2)
image(dist5$plan, axes=FALSE, main=pm5)
par(opar)
```

`rmvnorm` *Generate Random Samples from Multivariate Normal Distribution*

Description

In $\mathbf{R}^p$, random samples are drawn

$$X_1, X_2, \ldots, X_n \ \sim \ \mathcal{N}(\mu, \Sigma)$$

where $\mu \in \mathbf{R}^p$ is a mean vector and $\Sigma \in \mathrm{SPD}(p)$ is a positive definite covariance matrix.

Usage

```
rmvnorm(n = 1, mu, sigma)
```

Arguments

`n`	the number of samples to be generated.
`mu`	mean vector.
`sigma`	covariance matrix.

Value

either (1) a length-p vector ($n = 1$) or (2) an $(n \times p)$ matrix where rows are random samples.

Examples

```
#-------------------------------------------------------------------
#   Generate Random Data and Compare with Empirical Covariances
#
# In R^5 with zero mean and diagonal covariance,
# generate 100 and 200 observations and compute MLE covariance.
#-------------------------------------------------------------------
## GENERATE DATA
mymu  = rep(0,5)
mysig = diag(5)

## MLE FOR COVARIANCE
smat1 = stats::cov(rmvnorm(n=100, mymu, mysig))
smat2 = stats::cov(rmvnorm(n=200, mymu, mysig))

## VISUALIZE
opar <- par(no.readonly=TRUE)
par(mfrow=c(1,3), pty="s")
image(mysig[,5:1], axes=FALSE, main="true covariance")
image(smat1[,5:1], axes=FALSE, main="empirical cov with n=100")
image(smat2[,5:1], axes=FALSE, main="empirical cov with n=200")
par(opar)
```

---

spd.geometry *Supported Geometries on SPD Manifold*

Description

SPD manifold is a well-studied space in that there have been many geometries proposed on the space. For special functions on under SPD category, this function finds whether there exists a matching name that is currently supported in **Riemann**. If there is none, it will return an error message.

Usage

```
spd.geometry(geometry)
```

Arguments

geometry	name of supported geometries, including
	AIRM Affine-Invariant Riemannian Metric.
	LERM Log-Euclidean Riemannian Metric.
	Jeffrey Jeffrey's divergence.
	Stein Stein's metric.
	Wasserstein 2-Wasserstein geometry.

Value

a matching name in lower-case.

Examples

```
# it just returns a small-letter string.
mygeom = spd.geometry("stein")
```

`spd.pdist` *Pairwise Distance on SPD Manifold*

Description

Given N observations $X_1, X_2, \ldots, X_N$ in SPD manifold, compute pairwise distances among observations.

Usage

```
spd.pdist(spdobj, geometry, as.dist = FALSE)
```

Arguments

`spdobj`	a S3 `"riemdata"` class of SPD-valued data.
`geometry`	name of the geometry to be used. See `spd.geometry` for supported geometries.
`as.dist`	logical; if `TRUE`, it returns a `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

```
#-----------------------------------------------------------------
#                   Two Types of Covariances
#
#  group1 : perturbed from data by N(0,1) in R^3
#  group2 : perturbed from data by [sin(x); cos(x); sin(x)*cos(x)]
#-----------------------------------------------------------------
## GENERATE DATA
spd_mats = array(0,c(3,3,20))
for (i in 1:10){
  spd_mats[,,i] = stats::cov(matrix(rnorm(50*3), ncol=3))
}
for (j in 11:20){
  randvec = stats::rnorm(50, sd=3)
  randmat = cbind(sin(randvec), cos(randvec), sin(randvec)*cos(randvec))
```

```
  spd_mats[,,j] = stats::cov(randmat + matrix(rnorm(50*3, sd=0.1), ncol=3))
}

## WRAP IT AS SPD OBJECT
spd_obj = wrap.spd(spd_mats)

## COMPUTE PAIRWISE DISTANCES
#  Geometries are case-insensitive.
pdA = spd.pdist(spd_obj, "airM")
pdL = spd.pdist(spd_obj, "lErm")
pdJ = spd.pdist(spd_obj, "Jeffrey")
pdS = spd.pdist(spd_obj, "stEin")
pdW = spd.pdist(spd_obj, "wasserstein")

## VISUALIZE
opar <- par(no.readonly=TRUE)
par(mfrow=c(2,3), pty="s")
image(pdA, axes=FALSE, main="AIRM")
image(pdL, axes=FALSE, main="LERM")
image(pdJ, axes=FALSE, main="Jeffrey")
image(pdS, axes=FALSE, main="Stein")
image(pdW, axes=FALSE, main="Wasserstein")
par(opar)
```

`spd.wassbary` *Wasserstein Barycenter of SPD Matrices*

Description

Given N observations $X_1, X_2, \ldots, X_N$ in SPD manifold, compute the L_2-Wasserstein barycenter that minimizes

$$\sum_{n=1}^{N} \lambda_i \mathcal{W}_2(N(X), N(X_i))^2$$

where $N(X)$ denotes the zero-mean Gaussian measure with covariance X.

Usage

```
spd.wassbary(spdobj, weight = NULL, method = c("RU02", "AE16"), ...)
```

Arguments

`spdobj`	a S3 `"riemdata"` class of SPD-valued data of $(p \times p)$ matrices.
`weight`	weight of observations; if `NULL` it assumes equal weights, or a nonnegative length-N vector that sums to 1 should be given.
`method`	name of the algortihm to be used; one of the `"RU02"`, `"AE16"`.

... extra parameters including

maxiter maximum number of iterations to be run (default:20).

abstol tolerance level for stopping criterion (default: 1e-8).

Value

a $(p \times p)$ Wasserstein barycenter matrix.

Examples

```
#-----------------------------------------------------------------
#        Covariances from standard multivariate Gaussians.
#-----------------------------------------------------------------
## GENERATE DATA
ndata = 20
pdim  = 10
mydat = array(0,c(pdim,pdim,ndata))
for (i in 1:ndata){
  mydat[,,i] = stats::cov(matrix(rnorm(100*pdim), ncol=pdim))
}
myriem = wrap.spd(mydat)

## COMPUTE BY DIFFERENT ALGORITHMS
baryRU <- spd.wassbary(myriem, method="RU02")
baryAE <- spd.wassbary(myriem, method="AE16")

## VISUALIZE
opar <- par(no.readonly=TRUE)
par(mfrow=c(1,3), pty="s")
image(diag(pdim), axes=FALSE, main="True Covariance")
image(baryRU, axes=FALSE, main="by RU02")
image(baryAE, axes=FALSE, main="by AE16")
par(opar)
```

`sphere.convert` *Convert between Cartesian Coordinates and Geographic Coordinates*

Description

In geospatial data analysis, it is common to consider locations on the Earth as data. These locations, usually provided by latitude and longitude, are not directly applicable for spherical data analysis. We provide two functions - `sphere.geo2xyz` and `sphere.xyz2geo` - that convert geographic coordinates in longitude/latitude into a unit-norm vector on $\mathcal{S}^2$, and vice versa. As a convention, latitude and longitude are represented as *decimal degrees*.

Usage

```
sphere.geo2xyz(lat, lon)

sphere.xyz2geo(xyz)
```

Arguments

lat	latitude (in decimal degrees).
lon	longitude (in decimal degrees).
xyz	a unit-norm vector in $\mathcal{S}^2$.

Value

transformed data.

Examples

```
## EXAMPLE DATA WITH POPULATED US CITIES
data(cities)

## SELECT ALBUQUERQUE
geo = cities$coord[1,]
xyz = cities$cartesian[1,]

## CHECK TWO INPUT TYPES AND THEIR CONVERSIONS
sphere.geo2xyz(geo[1], geo[2])
sphere.xyz2geo(xyz)
```

---

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.

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 $\left\{^{\{X_1,X_2,\ldots,X_M}\right\}$ 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.

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

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")
```

---

splaplace *Spherical Laplace Distribution*

Description

This is a collection of tools for learning with spherical Laplace (SL) distribution on a $(p-1)$-dimensional sphere in $\mathbf{R}^p$ including sampling, density evaluation, and maximum likelihood estimation of the parameters. The SL distribution is characterized by the following density function,

$$f_{SL}(x;\mu,\sigma) = \frac{1}{C(\sigma)} \exp\left(-\frac{d(x,\mu)}{\sigma}\right)$$

for location and scale parameters μ and σ respectively.

Usage

```
dsplaplace(data, mu, sigma, log = FALSE)

rsplaplace(n, mu, sigma)

mle.splaplace(data, method = c("DE", "Optimize", "Newton"), ...)
```

Arguments

data	data vectors in form of either an $(n \times p)$ matrix or a length-n list. See wrap.sphere for descriptions on supported input types.
mu	a length-p unit-norm vector of location.
sigma	a scale parameter that is positive.
log	a logical; TRUE to return log-density, FALSE for densities without logarithm applied.
n	the number of samples to be generated.
method	an algorithm name for concentration parameter estimation. It should be one of "Newton", "Optimize", and "DE" (case-sensitive).
...	extra parameters for computations, including **maxiter** maximum number of iterations to be run (default:50). **eps** tolerance level for stopping criterion (default: 1e-6). **use.exact** a logical to use exact (TRUE) or approximate (FALSE) updating rules (default: FALSE).

Value

dsplaplace gives a vector of evaluated densities given samples. rsplaplace generates unit-norm vectors in $\mathbf{R}^p$ wrapped in a list. mle.splaplace computes MLEs and returns a list containing estimates of location (mu) and scale (sigma) parameters.

Examples

```
# ------------------------------------------------------------------
#          Example with Spherical Laplace Distribution
#
# Given a fixed set of parameters, generate samples and acquire MLEs.
# Especially, we will see the evolution of estimation accuracy.
# ------------------------------------------------------------------
## DEFAULT PARAMETERS
true.mu  = c(1,0,0,0,0)
true.sig = 1

## GENERATE A RANDOM SAMPLE OF SIZE N=1000
big.data = rsplaplace(1000, true.mu, true.sig)

## ITERATE FROM 50 TO 1000 by 10
idseq = seq(from=50, to=1000, by=10)
```

```
nseq  = length(idseq)

hist.mu  = rep(0, nseq)
hist.sig = rep(0, nseq)

for (i in 1:nseq){
  small.data = big.data[1:idseq[i]]             # data subsetting
  small.MLE  = mle.splaplace(small.data)        # compute MLE

  hist.mu[i]  = acos(sum(small.MLE$mu*true.mu)) # difference in mu
  hist.sig[i] = small.MLE$sigma
}

## VISUALIZE
opar <- par(no.readonly=TRUE)
par(mfrow=c(1,2))
plot(idseq, hist.mu,  "b", pch=19, cex=0.5,
     main="difference in location", xlab="sample size")
plot(idseq, hist.sig, "b", pch=19, cex=0.5,
     main="scale parameter", xlab="sample size")
abline(h=true.sig, lwd=2, col="red")
par(opar)
```

---

spnorm *Spherical Normal Distribution*

Description

We provide tools for an isotropic spherical normal (SN) distributions on a $(p-1)$-sphere in $\mathbf{R}^p$ for sampling, density evaluation, and maximum likelihood estimation of the parameters where the density is defined as

$$f_{SN}(x;\mu,\lambda) = \frac{1}{Z(\lambda)} \exp\left(-\frac{\lambda}{2} d^2(x,\mu)\right)$$

for location and concentration parameters μ and λ respectively and the normalizing constant $Z(\lambda)$.

Usage

```
dspnorm(data, mu, lambda, log = FALSE)

rspnorm(n, mu, lambda)

mle.spnorm(data, method = c("Newton", "Halley", "Optimize", "DE"), ...)
```

Arguments

data	data vectors in form of either an $(n \times p)$ matrix or a length-n list. See wrap.sphere for descriptions on supported input types.
mu	a length-p unit-norm vector of location.
lambda	a concentration parameter that is positive.
log	a logical; TRUE to return log-density, FALSE for densities without logarithm applied.
n	the number of samples to be generated.
method	an algorithm name for concentration parameter estimation. It should be one of "Newton","Halley","Optimize", and "DE" (case sensitive).
...	extra parameters for computations, including **maxiter** maximum number of iterations to be run (default:50). **eps** tolerance level for stopping criterion (default: 1e-5).

Value

dspnorm gives a vector of evaluated densities given samples. rspnorm generates unit-norm vectors in $\mathbf{R}^p$ wrapped in a list. mle.spnorm computes MLEs and returns a list containing estimates of location (mu) and concentration (lambda) parameters.

References

Hauberg S (2018). "Directional Statistics with the Spherical Normal Distribution." In *2018 21st International Conference on Information Fusion (FUSION)*, 704–711. ISBN 978-0-9964527-6-2.

You K, Suh C (2022). "Parameter Estimation and Model-Based Clustering with Spherical Normal Distribution on the Unit Hypersphere." *Computational Statistics and Data Analysis*, 107457. ISSN 01679473.

Examples

```
# ------------------------------------------------------------------
#          Example with Spherical Normal Distribution
#
# Given a fixed set of parameters, generate samples and acquire MLEs.
# Especially, we will see the evolution of estimation accuracy.
# ------------------------------------------------------------------
## DEFAULT PARAMETERS
true.mu  = c(1,0,0,0,0)
true.lbd = 5

## GENERATE DATA N=1000
big.data = rspnorm(1000, true.mu, true.lbd)

## ITERATE FROM 50 TO 1000 by 10
idseq = seq(from=50, to=1000, by=10)
nseq  = length(idseq)
```

```
hist.mu  = rep(0, nseq)
hist.lbd = rep(0, nseq)

for (i in 1:nseq){
  small.data = big.data[1:idseq[i]]          # data subsetting
  small.MLE  = mle.spnorm(small.data) # compute MLE

  hist.mu[i]  = acos(sum(small.MLE$mu*true.mu)) # difference in mu
  hist.lbd[i] = small.MLE$lambda
}

## VISUALIZE
opar <- par(no.readonly=TRUE)
par(mfrow=c(1,2))
plot(idseq, hist.mu,  "b", pch=19, cex=0.5, main="difference in location")
plot(idseq, hist.lbd, "b", pch=19, cex=0.5, main="concentration param")
abline(h=true.lbd, lwd=2, col="red")
par(opar)
```

`stiefel.optSA` *Simulated Annealing on Stiefel Manifold*

Description

Simulated Annealing is a black-box, derivative-free optimization algorithm that iterates via stochastic search in the neighborhood of current position. `stiefel.optSA` solves the following problem

$$\min_X f(X), \quad X \in St(p,k)$$

without any other auxiliary information such as gradient or hessian involved.

Usage

```
stiefel.optSA(func, p, k, ...)
```

Arguments

`func`	a function to be *minimized*.
`p`	dimension parameter as in $St(k,p)$.
`k`	dimension parameter as in $St(k,p)$.
`...`	extra parameters for SA algorithm including
	n.start number of runs; algorithm is executed `n.start` times (default: 5).
	stepsize size of random walk on each component (default: 0.1).
	maxiter maximum number of iterations for each run (default: 100).
	cooling triplet for cooling schedule. See the section for the usage.
	init.val if `NULL`, starts from a random point. Otherwise, a Stiefel matrix of size (p,k) should be provided for fixed starting point.
	print.progress a logical; if `TRUE`, it prints each iteration.

Value

a named list containing:

cost minimized function value.

solution a $(p \times k)$ matrix that attains the `cost`.

accfreq frequency of acceptance moves.

Examples

```
#-----------------------------------------------------------------
#               Optimization for Eigen-Decomposition
#
# Given (5x5) covariance matrix S, eigendecomposition is indeed
# an optimization problem cast on the stiefel manifold. Here,
# we are trying to find top 3 eigenvalues and compare.
#-----------------------------------------------------------------
## PREPARE
set.seed(121)                          # set seed
A = cov(matrix(rnorm(100*5), ncol=5)) # define covariance
myfunc <- function(p){                 # cost function to minimize
  return(sum(-diag(t(p)%*%A%*%p)))
}

## SOLVE THE OPTIMIZATION PROBLEM
Aout = stiefel.optSA(myfunc, p=5, k=3, n.start=40, maxiter=200)

## COMPUTE EIGENVALUES
#  1. USE SOLUTIONS TO THE ABOVE OPTIMIZATION
abase   = Aout$solution
eig3sol = sort(diag(t(abase)%*%A%*%abase), decreasing=TRUE)

#  2. USE BASIC 'EIGEN' FUNCTION
eig3dec = sort(eigen(A)$values, decreasing=TRUE)[1:3]

## VISUALIZE
opar <- par(no.readonly=TRUE)
yran = c(min(min(eig3sol),min(eig3dec))*0.95,
         max(max(eig3sol),max(eig3dec))*1.05)
plot(1:3, eig3sol, type="b", col="red",  pch=19, ylim=yran,
     xlab="index", ylab="eigenvalue", main="compare top 3 eigenvalues")
lines(1:3, eig3dec, type="b", col="blue", pch=19)
legend(1, 1, legend=c("optimization","decomposition"), col=c("red","blue"),
       lty=rep(1,2), pch=19)
par(opar)
```

`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.

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.

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

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.landmark` *Wrap Landmark Data on Shape Space*

Description

One of the frameworks used in shape space is to represent the data as landmarks. Each shape is a point set of k points in $\mathbf{R}^p$ where each point is a labeled object. We consider general landmarks in $p = 2, 3, \ldots$. Note that when $p > 2$, it is stratified space but we assume singularities do not exist or are omitted. The wrapper takes translation and scaling out from the data to make it *preshape* (centered, unit-norm). Also, for convenience, orthogonal Procrustes analysis is applied with the first observation being the reference so that all the other data are rotated to match the shape of the first.

Usage

```
wrap.landmark(input)
```

Arguments

input	data matrices to be wrapped as `riemdata` class. Following inputs are considered,
	array a $(k \times p \times n)$ array where each slice along 3rd dimension is a k-ad in $\mathbf{R}^p$.
	list a length-n list whose elements are k-ads.

Value

a named `riemdata` S3 object containing

data a list of preshapes in $\mathbf{R}^p$.

size size of each preshape.

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

References

Dryden IL, Mardia KV (2016). *Statistical shape analysis with applications in R*, Wiley series in probability and statistics, Second edition edition. John Wiley and Sons, Chichester, UK ; Hoboken, NJ. ISBN 978-1-119-07251-5 978-1-119-07250-8.

Examples

```
## USE 'GORILLA' DATA
data(gorilla)
riemobj = wrap.landmark(gorilla$male)
```

---

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 $(p \times p)$ SPD matrices.

size size of each SPD 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

∗ **basic**
riem.interp, 34
riem.interps, 35
riem.pdist, 57
riem.pdist2, 59
riem.wasserstein, 79
∗ **clustering**
riem.clrq, 24
riem.hclust, 33
riem.kmeans, 38
riem.kmeans18B, 40
riem.kmeanspp, 42
riem.kmedoids, 44
riem.nmshift, 56
riem.sc05Z, 66
riem.scNJW, 67
riem.scSM, 69
riem.scUL, 71
∗ **curve**
riem.distlp, 28
riem.dtw, 30
∗ **datasets**
cities, 5
ERP, 7
gorilla, 7
hands, 12
orbital, 21
passiflora, 22
∗ **data**
cities, 5
ERP, 7
gorilla, 7
hands, 12
orbital, 21
passiflora, 22
∗ **distribution**
acg, 3
macg, 15
splaplace, 88
spnorm, 90
∗ **grassmann**
grassmann.optmacg, 8
grassmann.runif, 10
grassmann.utest, 11
∗ **inference**
predict.m2skreg, 23
riem.fanova, 31
riem.m2skreg, 48
riem.m2skregCV, 50
riem.mean, 53
riem.median, 54
riem.test2bg14, 74
riem.test2wass, 76
∗ **learning**
riem.coreset18B, 26
riem.knn, 45
riem.rmml, 63
riem.seb, 72
∗ **spd**
spd.geometry, 82
spd.pdist, 83
spd.wassbary, 84
∗ **sphere**
moSL, 17
moSN, 19
sphere.convert, 85
sphere.runif, 86
sphere.utest, 87
∗ **stiefel**
stiefel.optSA, 92
stiefel.runif, 93
stiefel.utest, 95
∗ **utility**
density, 6
label, 13
loglkd, 14
rmvnorm, 81
∗ **visualization**

riem.isomap, 37
riem.kpca, 47
riem.mds, 51
riem.pga, 60
riem.phate, 61
riem.sammon, 64
riem.tsne, 78
∗ **wrapper**
wrap.correlation, 96
wrap.euclidean, 97
wrap.grassmann, 98
wrap.landmark, 99
wrap.multinomial, 100
wrap.rotation, 101
wrap.spd, 102
wrap.spdk, 103
wrap.sphere, 105
wrap.stiefel, 106

acg, 3, *16*

cities, 5

dacg *(*acg*)*, 3
density, 6
density.moSL *(*moSL*)*, 17
density.moSN *(*moSN*)*, 19
dmacg *(*macg*)*, 15
dsplaplace *(*splaplace*)*, 88
dspnorm *(*spnorm*)*, 90

ERP, 7

gorilla, 7
grassmann.optmacg, 8
grassmann.runif, 10
grassmann.utest, 11

hands, 12
hclust, *33*

label, 13
label.moSL *(*moSL*)*, 17
label.moSN *(*moSN*)*, 19
loglkd, 14
loglkd.moSL *(*moSL*)*, 17
loglkd.moSN *(*moSN*)*, 19

macg, 15
mle.acg *(*acg*)*, 3
mle.macg *(*macg*)*, 15
mle.splaplace *(*splaplace*)*, 88
mle.spnorm *(*spnorm*)*, 90
moSL, 17, *18*
moSN, 19, *20*

orbital, 21

pam, *44*
passiflora, 22
predict.m2skreg, 23

racg *(*acg*)*, 3
riem.clrq, 24
riem.coreset18B, 26, *41*
riem.distlp, 28
riem.dtw, 30
riem.fanova, 31
riem.fanovaP *(*riem.fanova*)*, 31
riem.hclust, 33
riem.interp, 34
riem.interps, 35
riem.isomap, 37
riem.kmeans, *25*, 38
riem.kmeans18B, 40
riem.kmeanspp, *39*, 42
riem.kmedoids, 44
riem.knn, 45
riem.kpca, 47
riem.m2skreg, *23, 24*, 48
riem.m2skregCV, 50
riem.mds, 51
riem.mean, 53
riem.median, 54
riem.nmshift, 56
riem.pdist, 57
riem.pdist2, 59
riem.pga, 60
riem.phate, 61
riem.rmml, 63
riem.sammon, 64
riem.sc05Z, 66
riem.scNJW, *66*, 67
riem.scSM, 69
riem.scUL, 71
riem.seb, 72
riem.test2bg14, 74
riem.test2wass, 76
riem.tsne, 78

riem.wasserstein, *76*, 79
rmacg *(*macg*)*, 15
rmvnorm, 81
rsplaplace *(*splaplace*)*, 88
rspnorm *(*spnorm*)*, 90

spd.geometry, 82, *83*
spd.pdist, 83
spd.wassbary, 84
sphere.convert, *5*, 85
sphere.geo2xyz *(*sphere.convert*)*, 85
sphere.runif, 86
sphere.utest, 87
sphere.xyz2geo *(*sphere.convert*)*, 85
splaplace, 88
spnorm, 90
stiefel.optSA, 92
stiefel.runif, *10*, 93
stiefel.utest, 95

wrap.correlation, 96
wrap.euclidean, 97
wrap.grassmann, *10*, *12*, 98
wrap.landmark, *8*, *13*, *23*, 99
wrap.multinomial, 100
wrap.rotation, 101
wrap.spd, *7*, 102
wrap.spdk, 103
wrap.sphere, *5*, *17, 18*, *20*, *22*, *86–89*, *91*, 105
wrap.stiefel, *94, 95*, 106