Package ‘maotai’

February 27, 2025

Type Package

Title Tools for Matrix Algebra, Optimization and Inference

Version 0.2.6

Description Matrix is an universal and sometimes primary object/unit in applied mathematics and statistics. We provide a number of algorithms for selected problems in optimization and statistical inference. For general exposition to the topic with focus on statistical context, see the book by Banerjee and Roy (2014, ISBN:9781420095388).

Encoding UTF-8

License MIT + file LICENSE

Suggests covr, igraph, testthat (>= 3.0.0)

Imports Matrix, Rcpp, Rdpack, RSpectra, Rtsne, RANN, cluster, labdsv, shapes, stats, utils, fastcluster, dbscan, pracma

LinkingTo Rcpp, RcppArmadillo, RcppDist

RdMacros Rdpack

RoxygenNote 7.3.2

URL `https://github.com/kisungyou/maotai`

BugReports `https://github.com/kisungyou/maotai/issues`

Config/testthat/edition 3

NeedsCompilation yes

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

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

Repository CRAN

Date/Publication 2025-02-27 16:40:02 UTC

Contents

`bmds` *Bayesian Multidimensional Scaling*

Description

A Bayesian formulation of classical Multidimensional Scaling is presented. Even though this method is based on MCMC sampling, we only return maximum a posterior (MAP) estimate that maximizes the posterior distribution. Due to its nature without any special tuning, increasing `mc.iter` requires much computation.

Usage

```
bmds(
  data,
  ndim = 2,
  par.a = 5,
  par.alpha = 0.5,
  par.step = 1,
  mc.iter = 8128,
  verbose = TRUE
)
```

Arguments

`data`	an $(n \times p)$ matrix whose rows are observations.
`ndim`	an integer-valued target dimension.
`par.a`	hyperparameter for conjugate prior on variance term, i.e., $\sigma^2 \sim IG(a, b)$. Note that b is chosen appropriately as in paper.
`par.alpha`	hyperparameter for conjugate prior on diagonal term, i.e., $\lambda_j \sim IG(\alpha, \beta_j)$. Note that β_j is chosen appropriately as in paper.
`par.step`	stepsize for random-walk, which is standard deviation of Gaussian proposal.
`mc.iter`	the number of MCMC iterations.
`verbose`	a logical; `TRUE` to show iterations, `FALSE` otherwise.

Value

a named list containing

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

stress discrepancy between embedded and origianl data as a measure of error.

References

Oh M, Raftery AE (2001). "Bayesian Multidimensional Scaling and Choice of Dimension." *Journal of the American Statistical Association*, **96**(455), 1031–1044.

Examples

```
## use simple example of iris dataset
data(iris)
idata = as.matrix(iris[,1:4])

## run Bayesian MDS
#  let's run 10 iterations only.
iris.cmds = cmds(idata, ndim=2)
iris.bmds = bmds(idata, ndim=2, mc.iter=5, par.step=(2.38^2))

## extract coordinates and class information
cx = iris.cmds$embed # embedded coordinates of CMDS
```

```
bx = iris.bmds$embed #                         BMDS
icol = iris[,5]      # class information

## visualize
opar <- par(no.readonly=TRUE)
par(mfrow=c(2,1))
mc = paste0("CMDS with STRESS=",round(iris.cmds$stress,4))
mb = paste0("BMDS with STRESS=",round(iris.bmds$stress,4))
plot(cx, col=icol,pch=19,main=mc)
plot(bx, col=icol,pch=19,main=mb)
par(opar)
```

boot.mblock *Generate Index for Moving Block Bootstrapping*

Description

Assuming data being dependent with cardinality `N`, `boot.mblock` returns a vector of index that is used for moving block bootstrapping.

Usage

```
boot.mblock(N, b = max(2, round(N/10)))
```

Arguments

`N`	the number of observations.
`b`	the size of a block to be drawn.

Value

a vector of length `N` for moving block bootstrap sampling.

References

Kunsch HR (1989). "The Jackknife and the Bootstrap for General Stationary Observations." *The Annals of Statistics*, **17**(3), 1217–1241.

Examples

```
## example : bootstrap confidence interval of mean and variances
vec.x = seq(from=0,to=10,length.out=100)
vec.y = sin(1.21*vec.x) + 2*cos(3.14*vec.x) + rnorm(100,sd=1.5)
data.mu  = mean(vec.y)
data.var = var(vec.y)

## apply moving block bootstrapping
```

```
nreps   = 50
vec.mu  = rep(0,nreps)
vec.var = rep(0,nreps)
for (i in 1:nreps){
   sample.id = boot.mblock(100, b=10)
   sample.y  = vec.y[sample.id]
   vec.mu[i]  = mean(sample.y)
   vec.var[i] = var(sample.y)
   print(paste("iteration ",i,"/",nreps," complete.", sep=""))
}

## visualize
opar <- par(no.readonly=TRUE)
par(mfrow=c(1,3), pty="s")
plot(vec.x, vec.y, type="l", main="1d signal")  # 1d signal
hist(vec.mu, main="mean CI", xlab="mu")         # mean
abline(v=data.mu, col="red", lwd=4)
hist(vec.var, main="variance CI", xlab="sigma") # variance
abline(v=data.var, col="blue", lwd=4)
par(opar)
```

`boot.stationary` *Generate Index for Stationary Bootstrapping*

Description

Assuming data being dependent with cardinality `N`, `boot.stationary` returns a vector of index that is used for stationary bootstrapping. To describe, starting points are drawn from uniform distribution over `1:N` and the size of each block is determined from geometric distribution with parameter p.

Usage

```
boot.stationary(N, p = 0.25)
```

Arguments

`N`	the number of observations.
`p`	parameter for geometric distribution with the size of each block.

Value

a vector of length `N` for moving block bootstrap sampling.

References

Politis DN, Romano JP (1994). “The Stationary Bootstrap.” *Journal of the American Statistical Association*, **89**(428), 1303. ISSN 01621459.

Examples

```
## example : bootstrap confidence interval of mean and variances
vec.x = seq(from=0,to=10,length.out=100)
vec.y = sin(1.21*vec.x) + 2*cos(3.14*vec.x) + rnorm(100,sd=1.5)
data.mu  = mean(vec.y)
data.var = var(vec.y)

## apply stationary bootstrapping
nreps   = 50
vec.mu  = rep(0,nreps)
vec.var = rep(0,nreps)
for (i in 1:nreps){
   sample.id = boot.stationary(100)
   sample.y  = vec.y[sample.id]
   vec.mu[i]  = mean(sample.y)
   vec.var[i] = var(sample.y)
   print(paste("iteration ",i,"/",nreps," complete.", sep=""))
}

## visualize
opar <- par(no.readonly=TRUE)
par(mfrow=c(1,3), pty="s")
plot(vec.x, vec.y, type="l", main="1d signal")  # 1d signal
hist(vec.mu, main="mean CI", xlab="mu")         # mean
abline(v=data.mu, col="red", lwd=4)
hist(vec.var, main="variance CI", xlab="sigma") # variance
abline(v=data.var, col="blue", lwd=4)
par(opar)
```

`cayleymenger` *Cayley-Menger Determinant*

Description

Cayley-Menger determinant is a formula of a n-dimensional simplex with respect to the squares of all pairwise distances of its vertices.

Usage

```
cayleymenger(data)
```

Arguments

`data`	an $(n \times p)$ matrix of row-stacked observations.

Value

a list containing

det determinant value.

vol volume attained from the determinant.

Examples

```
## USE 'IRIS' DATASET
data(iris)
X = as.matrix(iris[,1:4])

## COMPUTE CAYLEY-MENGER DETERMINANT
#  since k=4 < n=149, it should be zero.
cayleymenger(X)
```

---

checkdist *Check for Distance Matrix*

Description

This function checks whether the distance matrix $D := d_{ij} = d(x_i, x_j)$ satisfies three axioms to make itself a semimetric, which are (1) $d_{ii} = 0$, (2) $d_{ij} > 0$ for $i \neq j$, and (3) $d_{ij} = d_{ji}$.

Usage

```
checkdist(d)
```

Arguments

d	`"dist"` object or $(N \times N)$ matrix of pairwise distances.

Value

a logical; `TRUE` if it satisfies metric property, `FALSE` otherwise.

See Also

`checkmetric`

Examples

```
## Let's use L2 distance matrix of iris dataset
data(iris)
dx = as.matrix(stats::dist(iris[,1:4]))

# perturb d(i,j)
dy = dx
dy[1,2] <- dy[2,1] <- 10

# run the algorithm
checkdist(dx)
checkdist(dy)
```

`checkmetric` *Check for Metric Matrix*

Description

This function checks whether the distance matrix $D := d_{ij} = d(x_i, x_j)$ satisfies four axioms to make itself a semimetric, which are (1) $d_{ii} = 0$, (2) $d_{ij} > 0$ for $i \neq j$, (3) $d_{ij} = d_{ji}$, and (4) $d_{ij} \leq d_{ik} + d_{kj}$.

Usage

```
checkmetric(d)
```

Arguments

`d`	`"dist"` object or $(N \times N)$ matrix of pairwise distances.

Value

a logical; `TRUE` if it satisfies metric property, `FALSE` otherwise.

See Also

`checkdist`

Examples

```
## Let's use L2 distance matrix of iris dataset
data(iris)
dx = as.matrix(stats::dist(iris[,1:4]))

# perturb d(i,j)
dy = dx
dy[1,2] <- dy[2,1] <- 10
```

```
# run the algorithm
checkmetric(dx)
checkmetric(dy)
```

`cmds` *Classical Multidimensional Scaling*

Description

Classical multidimensional scaling aims at finding low-dimensional structure by preserving pairwise distances of data.

Usage

```
cmds(data, ndim = 2)
```

Arguments

`data`	an $(n \times p)$ matrix whose rows are observations.
`ndim`	an integer-valued target dimension.

Value

a named list containing

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

stress discrepancy between embedded and origianl data 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

```
## use simple example of iris dataset
data(iris)
idata = as.matrix(iris[,1:4])
icol  = as.factor(iris[,5])   # class information

## run Classical MDS
iris.cmds = cmds(idata, ndim=2)

## visualize
opar <- par(no.readonly=TRUE)
plot(iris.cmds$embed, col=icol,
     main=paste0("STRESS=",round(iris.cmds$stress,4)))
par(opar)
```

cov2corr *Convert Covariance into Correlation Matrix*

Description

Given a covariance matrix, return a correlation matrix that has unit diagonals. We strictly impose (and check) whether the given input is a symmetric matrix of full-rank.

Usage

```
cov2corr(mat)
```

Arguments

`mat`	a $(p \times p)$ covariance matrix.

Value

a $(p \times p)$ correlation matrix.

Examples

```
# generate an empirical covariance scaled
prep_mat = stats::cov(matrix(rnorm(100*10),ncol=10))
prep_vec = diag(as.vector(stats::runif(10, max=5)))
prep_cov = prep_vec%*%prep_mat%*%prep_vec

# compute correlation matrix
prep_cor = cov2corr(prep_cov)

# visualize
opar <- par(no.readonly=TRUE)
par(mfrow=c(1,2), pty="s")
image(prep_cov, axes=FALSE, main="covariance")
image(prep_cor, axes=FALSE, main="correlation")
par(opar)
```

cov2pcorr *Convert Covariance into Partial Correlation Matrix*

Description

Given a covariance matrix, return a partial correlation matrix that has unit diagonals. We strictly impose (and check) whether the given input is a symmetric matrix of full-rank.

Usage

```
cov2pcorr(mat)
```

Arguments

`mat`	a $(p \times p)$ covariance matrix.

Value

a $(p \times p)$ partial correlation matrix.

Examples

```
# generate an empirical covariance scaled
prep_mat = stats::cov(matrix(rnorm(100*10),ncol=10))
prep_vec = diag(as.vector(stats::runif(10, max=5)))
prep_cov = prep_vec%*%prep_mat%*%prep_vec

# compute correlation and partial correlation matrices
prep_cor = cov2corr(prep_cov)
prep_par = cov2pcorr(prep_cov)

# visualize
opar <- par(no.readonly=TRUE)
par(mfrow=c(1,3), pty="s")
image(prep_cov, axes=FALSE, main="covariance")
image(prep_cor, axes=FALSE, main="correlation")
image(prep_par, axes=FALSE, main="partial correlation")
par(opar)
```

`dpmeans` *DP-means Algorithm for Clustering Euclidean Data*

Description

DP-means is a nonparametric clustering method motivated by DP mixture model in that the number of clusters is determined by a parameter λ. The larger the λ value is, the smaller the number of clusters is attained. In addition to the original paper, we added an option to randomly permute an order of updating for each observation's membership as a common heuristic in the literature of cluster analysis.

Usage

```
dpmeans(
  data,
  lambda = 1,
  maxiter = 1234,
  abstol = 1e-06,
  permute.order = FALSE
)
```

Arguments

`data`	an $(n \times p)$ data matrix for each row being an observation.
`lambda`	a threshold to define a new cluster.
`maxiter`	maximum number of iterations.
`abstol`	stopping criterion
`permute.order`	a logical; `TRUE` if random order for permutation is used, `FALSE` otherwise.

Value

a named list containing

cluster an $(n \times ndim)$ matrix whose rows are embedded observations.

centers a list containing information for out-of-sample prediction.

References

Kulis B, Jordan MI (2012). "Revisiting K-Means: New Algorithms via Bayesian Nonparametrics." In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, ICML'12, 1131–1138. ISBN 978-1-4503-1285-1.

Examples

```
## define data matrix of two clusters
x1  = matrix(rnorm(50*3,mean= 2), ncol=3)
x2  = matrix(rnorm(50*3,mean=-2), ncol=3)
X   = rbind(x1,x2)
lab = c(rep(1,50),rep(2,50))

## run dpmeans with several lambda values
solA <- dpmeans(X, lambda= 5)$cluster
solB <- dpmeans(X, lambda=10)$cluster
solC <- dpmeans(X, lambda=20)$cluster

## visualize the results
opar <- par(no.readonly=TRUE)
par(mfrow=c(1,4), pty="s")
plot(X,col=lab,  pch=19, cex=.8, main="True", xlab="x", ylab="y")
plot(X,col=solA, pch=19, cex=.8, main="dpmeans lbd=5", xlab="x", ylab="y")
plot(X,col=solB, pch=19, cex=.8, main="dpmeans lbd=10", xlab="x", ylab="y")
plot(X,col=solC, pch=19, cex=.8, main="dpmeans lbd=20", xlab="x", ylab="y")
par(opar)


## let's find variations by permuting orders of update
## used setting : lambda=20, we will 8 runs
sol8 <- list()
for (i in 1:8){
  sol8[[i]] = dpmeans(X, lambda=20, permute.order=TRUE)$cluster
}

## let's visualize
vpar <- par(no.readonly=TRUE)
par(mfrow=c(2,4), pty="s")
for (i in 1:8){
  pm = paste("permute no.",i,sep="")
  plot(X,col=sol8[[i]], pch=19, cex=.8, main=pm, xlab="x", ylab="y")
}
par(vpar)
```

`ecdfdist` *Distance Measures between Multiple Empirical Cumulative Distribution Functions*

Description

We measure distance between two empirical cumulative distribution functions (ECDF). For simplicity, we only take an input of `ecdf` objects from **stats** package.

Usage

```
ecdfdist(elist, method = c("KS", "Lp", "Wasserstein"), p = 2, as.dist = FALSE)
```

Arguments

`elist`	a length N list of `ecdf` objects.
`method`	name of the distance/dissimilarity measure. Case insensitive.
`p`	exponent for `Lp` or `Wasserstein` distance.
`as.dist`	a logical; `TRUE` to return `dist` object, `FALSE` to return an $(N \times N)$ symmetric matrix of pairwise distances.

Value

either `dist` object of an $(N \times N)$ symmetric matrix of pairwise distances by `as.dist` argument.

See Also

`ecdf`

Examples

```
## toy example : 10 of random and uniform distributions
mylist = list()
for (i in 1:10){
  mylist[[i]] = stats::ecdf(stats::rnorm(50, sd=2))
}
for (i in 11:20){
  mylist[[i]] = stats::ecdf(stats::runif(50, min=-5))
}

## compute Kolmogorov-Smirnov distance
dm = ecdfdist(mylist, method="KS")

## visualize
mks  =" KS distances of 2 Types"
opar = par(no.readonly=TRUE)
par(pty="s")
image(dm[,nrow(dm):1], axes=FALSE, main=mks)
par(opar)
```

---

ecdfdist2 *Pairwise Measures for Two Sets of Empirical CDFs*

Description

We measure distance between two sets of empirical cumulative distribution functions (ECDF). For simplicity, we only take an input of `ecdf` objects from **stats** package.

Usage

```
ecdfdist2(elist1, elist2, method = c("KS", "Lp", "Wasserstein"), p = 2)
```

Arguments

`elist1`	a length M list of `ecdf` objects.
`elist2`	a length N list of `ecdf` objects.
`method`	name of the distance/dissimilarity measure. Case insensitive.
`p`	exponent for `Lp` or `Wasserstein` distance.

Value

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

See Also

`ecdf` `ecdfdist`

Examples

```
## toy example
#  first list : 10 of random and uniform distributions
mylist1 = list()
for (i in 1:10){ mylist1[[i]] = stats::ecdf(stats::rnorm(50, sd=2))}
for (i in 11:20){mylist1[[i]] = stats::ecdf(stats::runif(50, min=-5))}

#  second list : 15 uniform and random distributions
mylist2 = list()
for (i in 1:15){ mylist2[[i]] = stats::ecdf(stats::runif(50, min=-5))}
for (i in 16:30){mylist2[[i]] = stats::ecdf(stats::rnorm(50, sd=2))}

## compute Kolmogorov-Smirnov distance
dm2ks = ecdfdist2(mylist1, mylist2, method="KS")
dm2lp = ecdfdist2(mylist1, mylist2, method="lp")
dm2wa = ecdfdist2(mylist1, mylist2, method="wasserstein")
nrs   = nrow(dm2ks)

## visualize
opar = par(no.readonly=TRUE)
par(mfrow=c(1,3), pty="s")
image(dm2ks[,nrs:1], axes=FALSE, main="Kolmogorov-Smirnov")
image(dm2lp[,nrs:1], axes=FALSE, main="L2")
image(dm2wa[,nrs:1], axes=FALSE, main="Wasserstein")
par(opar)
```

---

epmeans   *EP-means Algorithm for Clustering Empirical Distributions*

Description

EP-means is a variant of k-means algorithm adapted to cluster multiple empirical cumulative distribution functions under metric structure induced by Earth Mover's Distance.

Usage

```
epmeans(elist, k = 2)
```

Arguments

elist	a length N list of either vector or ecdf objects.
k	the number of clusters.

Value

a named list containing

cluster an integer vector indicating the cluster to which each ecdf is allocated.

centers a length k list of centroid ecdf objects.

References

Henderson K, Gallagher B, Eliassi-Rad T (2015). “EP-MEANS: An Efficient Nonparametric Clustering of Empirical Probability Distributions.” In *Proceedings of the 30th Annual ACM Symposium on Applied Computing - SAC ’15*, 893–900. ISBN 978-1-4503-3196-8.

Examples

```
## two sets of 1d samples, 10 each and add some noise
#    set 1 : mixture of two gaussians
#    set 2 : single gamma distribution

# generate data
elist = list()
for (i in 1:10){
   elist[[i]] = stats::ecdf(c(rnorm(100, mean=-2), rnorm(50, mean=2)))
}
for (j in 11:20){
   elist[[j]] = stats::ecdf(rgamma(100,1) + rnorm(100, sd=sqrt(0.5)))
}

# run EP-means with k clusters
# change the value below to see different settings
myk   = 2
epout = epmeans(elist, k=myk)

# visualize
opar = par(no.readonly=TRUE)
par(mfrow=c(1,myk))
for (k in 1:myk){
  idk = which(epout$cluster==k)
  for (i in 1:length(idk)){
    if (i<2){
      pm = paste("class ",k," (size=",length(idk),")",sep="")
      plot(elist[[idk[i]]], verticals=TRUE, lwd=0.25, do.points=FALSE, main=pm)
    } else {
      plot(elist[[idk[i]]], add=TRUE, verticals=TRUE, lwd=0.25, do.points=FALSE)
    }
   plot(epout$centers[[k]], add=TRUE, verticals=TRUE, lwd=2, col="red", do.points=FALSE)
  }
}
par(opar)
```

---

kmeanspp *K-Means++ Clustering Algorithm*

Description

k-means++ algorithm is known to be a smart, careful initialization technique. It is originally intended to return a set of k points as initial centers though it can still be used as a rough clustering algorithm by assigning points to the nearest points.

Usage

```
kmeanspp(data, k = 2)
```

Arguments

`data`	an $(n \times p)$ matrix whose rows are observations.
`k`	the number of clusters.

Value

a length-n vector of class labels.

References

Arthur D, Vassilvitskii S (2007). “K-Means++: The Advantages of Careful Seeding.” In *In Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms*.

Examples

```
## use simple example of iris dataset
data(iris)
mydata = as.matrix(iris[,1:4])
mycol  = as.factor(iris[,5])

## find the low-dimensional embedding for visualization
my2d = cmds(mydata, ndim=2)$embed

## apply 'kmeanspp' with different numbers of k's.
k2 = kmeanspp(mydata, k=2)
k3 = kmeanspp(mydata, k=3)
k4 = kmeanspp(mydata, k=4)
k5 = kmeanspp(mydata, k=5)
k6 = kmeanspp(mydata, k=6)

## visualize
opar <- par(no.readonly=TRUE)
par(mfrow=c(2,3))
plot(my2d, col=k2, main="k=2", pch=19, cex=0.5)
plot(my2d, col=k3, main="k=3", pch=19, cex=0.5)
plot(my2d, col=k4, main="k=4", pch=19, cex=0.5)
plot(my2d, col=k5, main="k=5", pch=19, cex=0.5)
plot(my2d, col=k6, main="k=6", pch=19, cex=0.5)
plot(my2d, col=mycol, main="true cluster", pch=19, cex=0.5)
par(opar)
```

lgpa *Large-scale Generalized Procrustes Analysis*

Description

We modify generalized Procrustes analysis for large-scale data by first setting a subset of anchor points and applying the attained transformation to the rest data. If `sub.id` is a vector `1:dim(x)[1]`, it uses all observations as anchor points, reducing to the conventional generalized Procrustes analysis.

Usage

```
lgpa(x, sub.id = 1:(dim(x)[1]), scale = TRUE, reflect = FALSE)
```

Arguments

`x`	a $(k \times m \times n)$ 3d array, where k is the number of points, m the number of dimensions, and n the number of samples.
`sub.id`	a vector of indices for defining anchor points.
`scale`	a logical; `TRUE` if scaling is applied, `FALSE` otherwise.
`reflect`	a logical; `TRUE` if reflection is required, `FALSE` otherwise.

Value

a $(k \times m \times n)$ 3d array of aligned samples.

Author(s)

Kisung You

References

Goodall C (1991). "Procrustes Methods in the Statistical Analysis of Shape." *Journal of the Royal Statistical Society. Series B (Methodological)*, **53**(2), 285–339. ISSN 00359246.

Examples

```
## Not run:
## This should be run if you have 'shapes' package installed.
library(shapes)
data(gorf.dat)

## apply anchor-based method and original procGPA
out.proc = shapes::procGPA(gorf.dat, scale=TRUE)$rotated # procGPA from shapes package
out.anc4 = lgpa(gorf.dat, sub.id=c(1,4,9,7), scale=TRUE) # use 4 points
out.anc7 = lgpa(gorf.dat, sub.id=1:7, scale=TRUE)       # use all but 1 point as anchors

## visualize
```

```
opar = par(no.readonly=TRUE)
par(mfrow=c(3,4), pty="s")
plot(out.proc[,,1], main="procGPA No.1", pch=18)
plot(out.proc[,,2], main="procGPA No.2", pch=18)
plot(out.proc[,,3], main="procGPA No.3", pch=18)
plot(out.proc[,,4], main="procGPA No.4", pch=18)
plot(out.anc4[,,1], main="4 Anchors No.1", pch=18, col="blue")
plot(out.anc4[,,2], main="4 Anchors No.2", pch=18, col="blue")
plot(out.anc4[,,3], main="4 Anchors No.3", pch=18, col="blue")
plot(out.anc4[,,4], main="4 Anchors No.4", pch=18, col="blue")
plot(out.anc7[,,1], main="7 Anchors No.1", pch=18, col="red")
plot(out.anc7[,,2], main="7 Anchors No.2", pch=18, col="red")
plot(out.anc7[,,3], main="7 Anchors No.3", pch=18, col="red")
plot(out.anc7[,,4], main="7 Anchors No.4", pch=18, col="red")
par(opar)

## End(Not run)
```

`lyapunov` *Solve Lyapunov Equation*

Description

The Lyapunov equation is of form

$$AX + XA^\top = Q$$

where A and Q are square matrices of same size. Above form is also known as *continuous* form. This is a wrapper of `armadillo`'s `sylvester` function.

Usage

```
lyapunov(A, Q)
```

Arguments

`A`	a $(p \times p)$ matrix as above.
`Q`	a $(p \times p)$ matrix as above.

Value

a solution matrix X of size $(p \times p)$.

References

Sanderson C, Curtin R (2016). “Armadillo: A Template-Based C++ Library for Linear Algebra.” *The Journal of Open Source Software*, **1**(2), 26.

Eddelbuettel D, Sanderson C (2014). “RcppArmadillo: Accelerating R with High-Performance C++ Linear Algebra.” *Computational Statistics and Data Analysis*, **71**, 1054–1063.

Examples

```
## simulated example
#  generate square matrices
A = matrix(rnorm(25),nrow=5)
X = matrix(rnorm(25),nrow=5)
Q = A%*%X + X%*%t(A)

#  solve using 'lyapunov' function
solX = lyapunov(A,Q)
## Not run:
pm1 = "* Experiment with Lyapunov Solver"
pm2 = paste("* Absolute Error  : ",norm(solX-X,"f"),sep="")
pm3 = paste("* Relative Error  : ",norm(solX-X,"f")/norm(X,"f"),sep="")
cat(paste(pm1,"\n",pm2,"\n",pm3,sep=""))

## End(Not run)
```

`matderiv` *Numerical Approximation to Gradient of a Function with Matrix Argument*

Description

For a given function $f : \mathbf{R}^{n\times p} \rightarrow \mathbf{R}$, we use finite difference scheme that approximates a gradient at a given point x. In Riemannian optimization, this can be used as a proxy for ambient gradient. Use with care since it may accumulate numerical error.

Usage

```
matderiv(fn, x, h = 0.001)
```

Arguments

`fn`	a function that takes a matrix of size $(n \times p)$ and returns a scalar value.
`x`	an $(n \times p)$ matrix where the gradient is to be computed.
`h`	step size for centered difference scheme.

Value

an approximate numerical gradient matrix of size $(n \times p)$.

References

Kincaid D, Cheney EW (2009). *Numerical Analysis: Mathematics of Scientific Computing*, number 2 in Pure and Applied Undergraduate Texts, 3. ed edition. American Mathematical Society, Providence, RI.

Examples

```
## function f(X) = <a,Xb> for two vectors 'a' and 'b'
#  derivative w.r.t X is ab'
#  take an example of (5x5) symmetric positive definite matrix

#  problem settings
a   <- rnorm(5)
b   <- rnorm(5)
ftn <- function(X){
  return(sum(as.vector(X%*%b)*a))
}       # function to be taken derivative
myX <- matrix(rnorm(25),nrow=5)  # point where derivative is evaluated
myX <- myX%*%t(myX)

# main computation
sol.true <- base::outer(a,b)
sol.num1 <- matderiv(ftn, myX, h=1e-1) # step size : 1e-1
sol.num2 <- matderiv(ftn, myX, h=1e-5) #             1e-3
sol.num3 <- matderiv(ftn, myX, h=1e-9) #             1e-5

## visualize/print the results
expar = par(no.readonly=TRUE)
par(mfrow=c(2,2),pty="s")
image(sol.true, main="true solution")
image(sol.num1, main="h=1e-1")
image(sol.num2, main="h=1e-5")
image(sol.num3, main="h=1e-9")
par(expar)


ntrue = norm(sol.true,"f")
cat('* Relative Errors in Frobenius Norm ')
cat(paste("*  h=1e-1   : ",norm(sol.true-sol.num1,"f")/ntrue,sep=""))
cat(paste("*  h=1e-5   : ",norm(sol.true-sol.num2,"f")/ntrue,sep=""))
cat(paste("*  h=1e-9   : ",norm(sol.true-sol.num3,"f")/ntrue,sep=""))
```

`metricdepth` *Metric Depth*

Description

Compute the metric depth proposed by Geenens et al. (2023), which is one generalization of statistical depth function onto the arbitrary metric space. Our implementation assumes that given the multivariate data it computes the (empirical) depth for all observations using under the Euclidean regime.

Usage

```
metricdepth(data)
```

Arguments

`data`	an $(n \times p)$ matrix whose rows are observations.

Value

a length-n vector of empirical metric depth values.

References

Geenens G, Nieto-Reyes A, Francisci G (2023). “Statistical Depth in Abstract Metric Spaces.” *Statistics and Computing*, **33**(2), 46. ISSN 0960-3174, 1573-1375.

Examples

```
## Not run:
## use simple example of iris dataset
data(iris)
X <- as.matrix(iris[,1:4])
y <- as.factor(iris[,5])

## compute the metric depth
mdX <- metricdepth(X)

## visualize
#  2-d embedding for plotting by MDS
X2d <- maotai::cmds(X, ndim=2)$embed

# get a color code for the metric depth
pal = colorRampPalette(c("yellow","red"))

# draw
opar <- par(no.readonly=TRUE)
par(mfrow=c(1,2), pty="s")
plot(X2d, pch=19, main="by class", xlab="", ylab="", col=y)
plot(X2d, pch=19, main="by depth", xlab="", ylab="", col=pal(150)[order(mdX)])
legend("bottomright", col=pal(2), pch=19, legend=round(range(mdX), 2))
par(opar)

## End(Not run)
```

`mmd2test` *Kernel Two-sample Test with Maximum Mean Discrepancy*

Description

Maximum Mean Discrepancy (MMD) as a measure of discrepancy between samples is employed as a test statistic for two-sample hypothesis test of equal distributions. Kernel matrix K is a symmetric square matrix that is positive semidefinite.

Usage

```
mmd2test(K, label, method = c("b", "u"), mc.iter = 999)
```

Arguments

`K`	kernel matrix or an object of `kernelMatrix` class from **kernlab** package.
`label`	label vector of class indices.
`method`	type of estimator to be used. `"b"` for biased and `"u"` for unbiased estimator of MMD.
`mc.iter`	the number of Monte Carlo resampling iterations.

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 kernel matrix.

References

Gretton A, Borgwardt KM, Rasch MJ, Schölkopf B, Smola A (2012). "A Kernel Two-Sample Test." *J. Mach. Learn. Res.*, **13**, 723–773. ISSN 1532-4435.

Examples

```
## small test for CRAN submission
dat1 <- matrix(rnorm(60, mean= 1), ncol=2) # group 1 : 30 obs of mean  1
dat2 <- matrix(rnorm(50, mean=-1), ncol=2) # group 2 : 25 obs of mean -1

dmat <- as.matrix(dist(rbind(dat1, dat2)))  # Euclidean distance matrix
kmat <- exp(-(dmat^2))                      # build a gaussian kernel matrix
lab  <- c(rep(1,30), rep(2,25))             # corresponding label

mmd2test(kmat, lab)                         # run the code !

## Not run:
## WARNING: computationally heavy.
#  Let's compute empirical Type 1 error at alpha=0.05
niter = 496
pvals1 = rep(0,niter)
pvals2 = rep(0,niter)
for (i in 1:niter){
  dat = matrix(rnorm(200),ncol=2)
  lab = c(rep(1,50), rep(2,50))
  lbd = 0.1
  kmat = exp(-lbd*(as.matrix(dist(dat))^2))
  pvals1[i] = mmd2test(kmat, lab, method="b")$p.value
  pvals2[i] = mmd2test(kmat, lab, method="u")$p.value
  print(paste("iteration ",i," complete..",sep=""))
}

#  Visualize the above at multiple significance levels
alphas = seq(from=0.001, to=0.999, length.out=100)
errors1 = rep(0,100)
errors2 = rep(0,100)
for (i in 1:100){
   errors1[i] = sum(pvals1<=alphas[i])/niter
   errors2[i] = sum(pvals2<=alphas[i])/niter
}

opar <- par(no.readonly=TRUE)
par(mfrow=c(1,2), pty="s")
plot(alphas, errors1, "b", main="Biased Estimator Error",
     xlab="alpha", ylab="error", cex=0.5)
abline(a=0,b=1, lwd=1.5, col="red")
plot(alphas, errors2, "b", main="Unbiased Estimator Error",
     xlab="alpha", ylab="error", cex=0.5)
abline(a=0,b=1, lwd=1.5, col="blue")
par(opar)

## End(Not run)
```

movMF_convert *Convert 'movMF' object*

Description

Given an output from the movMF package's movMF function, convert them into the standard mixture parameter format.

Usage

```
movMF_convert(movMF_object)
```

Arguments

`movMF_object`	a movMF object of K components in d dimensions.

Value

a named list containing

means a $(K \times d)$ matrix of means

concentrations a K vector of concentrations

weights a K vector of weights

`movMF_info` *Extract meaningful information from the von Mises-Fisher mixture model*

Description

Given a mixture of von Mises-Fisher distributions, this function computes several related quantities of the data on the unit hypersphere with respect to the specified model.

Usage

```
movMF_info(data, means, concentrations, weights)
```

Arguments

`data`	an $(n \times d)$ data matrix.
`means`	an $(k \times d)$ matrix of means.
`concentrations`	a vector of length k of concentration parameters.
`weights`	a vector of length k of mixing weights.

Value

a named list containing

densities a vector of length n of the densities of the data points.

clustering a vector of length n of the hard clustering results.

loglkd the log-likelihood of the data.

AIC the Akaike information criterion.

BIC the Bayesian information criterion.

`movMF_reduce_greedy` *von Mises-Fisher mixture model reduction - Greedy method*

Description

When given parameters of the von Mises-Fisher mixture model, this function aims at mixture model reduction using a greedy method.

Usage

```
movMF_reduce_greedy(means, concentrations, weights, target.num = 2)
```

Arguments

`means`	a $(K \times p)$ matrix of means of the von Mises-Fisher components.
`concentrations`	a K vector of concentrations of the von Mises-Fisher components.
`weights`	a K vector of weights of the von Mises-Fisher components.
`target.num`	a desired number of components after reduction. Default is 2.

Value

a named list of the reduced mixture model containing

means a $(\texttt{target.num} \times p)$ matrix of means of the von Mises-Fisher components.

concentrations a `target.num` vector of concentrations of the von Mises-Fisher components.

weights a `target.num` vector of weights of the von Mises-Fisher components.

`movMF_reduce_partitional` *von Mises-Fisher mixture model reduction - partitional method*

Description

When given parameters of the von Mises-Fisher mixture model, this function aims at mixture model reduction using a partitional method.

Usage

```
movMF_reduce_partitional(
  means,
  concentrations,
  weights,
  target.num = 2,
  method = c("hclust", "kmedoids")
)
```

Arguments

`means`	a $(K \times p)$ matrix of means of the von Mises-Fisher components.
`concentrations`	a K vector of concentrations of the von Mises-Fisher components.
`weights`	a K vector of weights of the von Mises-Fisher components.
`target.num`	a desired number of components after reduction. Default is 2.
`method`	a clustering method to be used. Default is "hclust".

Value

a named list of the reduced mixture model containing

means a ($\texttt{target.num} \times p$) matrix of means of the von Mises-Fisher components.

concentrations a `target.num` vector of concentrations of the von Mises-Fisher components.

weights a `target.num` vector of weights of the von Mises-Fisher components.

`nef` *Negative Eigenfraction*

Description

Negative Eigenfraction (NEF) is a measure of distortion for the data whether they are lying in Euclidean manner or not. When the value is exactly 0, it means the data is Euclidean. On the other hand, when NEF is far away from 0, it means not Euclidean. The concept of NEF is closely related to the definiteness of a Gram matrix.

Usage

```
nef(data)
```

Arguments

`data`	an $(n \times p)$ matrix whose rows are observations.

Value

a nonnegative NEF value.

References

Pękalska E, Harol A, Duin RPW, Spillmann B, Bunke H (2006). “Non-Euclidean or Non-Metric Measures Can Be Informative.” In Yeung D, Kwok JT, Fred A, Roli F, de Ridder D (eds.), *Structural, Syntactic, and Statistical Pattern Recognition*, 871–880. ISBN 978-3-540-37241-7.

Examples

```
## use simple example of iris dataset
data(iris)
mydat = as.matrix(iris[,1:4])

## calculate NEF
nef(mydat)
```

nem *Negative Eigenvalue Magnitude*

Description

Negative Eigenvalue Magnitude (NEM) is a measure of distortion for the data whether they are lying in Euclidean manner or not. When the value is exactly 0, it means the data is Euclidean. On the other hand, when NEM is far away from 0, it means not Euclidean. The concept of NEM is closely related to the definiteness of a Gram matrix.

Usage

```
nem(data)
```

Arguments

`data`	an $(n \times p)$ matrix whose rows are observations.

Value

a nonnegative NEM value.

References

Pękalska E, Harol A, Duin RPW, Spillmann B, Bunke H (2006). "Non-Euclidean or Non-Metric Measures Can Be Informative." In Yeung D, Kwok JT, Fred A, Roli F, de Ridder D (eds.), *Structural, Syntactic, and Statistical Pattern Recognition*, 871–880. ISBN 978-3-540-37241-7.

Examples

```
## use simple example of iris dataset
data(iris)
mydat = as.matrix(iris[,1:4])

## calculate NEM
nem(mydat)
```

pdeterminant *Calculate the Pseudo-Determinant of a Matrix*

Description

When a given square matrix A is rank deficient, determinant is zero. Still, we can compute the pseudo-determinant by multiplying all non-zero eigenvalues. Since thresholding to determine near-zero eigenvalues is subjective, we implemented the function as of original limit problem. When matrix is non-singular, it coincides with traditional determinant.

Usage

```
pdeterminant(A)
```

Arguments

A	a square matrix whose pseudo-determinant be computed.

Value

a scalar value for computed pseudo-determinant.

References

Holbrook A (2018). "Differentiating the Pseudo Determinant." *Linear Algebra and its Applications*, **548**, 293–304.

Examples

```
## show the convergence of pseudo-determinant
#  settings
n = 10
A = cov(matrix(rnorm(5*n),ncol=n))   # (n x n) matrix
k = as.double(Matrix::rankMatrix(A)) # rank of A

# iterative computation
ntry = 11
del.vec = exp(-(1:ntry))
det.vec = rep(0,ntry)
for (i in 1:ntry){
  del = del.vec[i]
  det.vec[i] = det(A+del*diag(n))/(del^(n-k))
}

# visualize the results
opar <- par(no.readonly=TRUE)
plot(1:ntry, det.vec, main=paste("true rank is ",k," out of ",n,sep=""),"b", xlab="iterations")
abline(h=pdeterminant(A),col="red",lwd=1.2)
par(opar)
```

rotationS2 *Compute a Rotation on the 2-dimensional Sphere*

Description

A vector of unit norm is an element on the hypersphere. When two unit-norm vectors x and y in 3-dimensional space are given, this function computes a rotation matrix Q on the 2-dimensional sphere such that

$$y = Qx$$

.

Usage

```
rotationS2(x, y)
```

Arguments

`x`	a length-3 vector. If $\|x\| \neq 1$, normalization is applied.
`y`	a length-3 vector. If $\|y\| \neq 1$, normalization is applied.

Value

a (3×3) rotation matrix.

Examples

```
## generate two data points
#  one randomly and another on the north pole
x = stats::rnorm(3)
x = x/sqrt(sum(x^2))
y = c(0,0,1)

## compute the rotation
Q = rotationS2(x,y)

## compare
Qx = as.vector(Q%*%x)

## print
printmat = rbind(Qx, y)
rownames(printmat) = c("rotated:", "target:")
print(printmat)
```

`shortestpath` *Find Shortest Path using Floyd-Warshall Algorithm*

Description

This is a fast implementation of Floyd-Warshall algorithm to find the shortest path in a pairwise sense using `RcppArmadillo`. A logical input is also accepted. The given matrix should contain pairwise distance values $d_{i,j}$ where 0 means there exists no path for node i and j.

Usage

```
shortestpath(dist)
```

Arguments

`dist`	either an $(n \times n)$ matrix or a `dist` class object.

Value

an $(n \times n)$ matrix containing pairwise shortest path length.

References

Floyd RW (1962). "Algorithm 97: Shortest Path." *Communications of the ACM*, **5**(6), 345.

Warshall S (1962). "A Theorem on Boolean Matrices." *Journal of the ACM*, **9**(1), 11–12.

Examples

```
## simple example : a ring graph
#  edges exist for pairs
A = array(0,c(10,10))
for (i in 1:9){
  A[i,i+1] = 1
  A[i+1,i] = 1
}
A[10,1] <- A[1,10] <- 1

# compute shortest-path and show the matrix
sdA <- shortestpath(A)

# visualize
opar <- par(no.readonly=TRUE)
par(pty="s")
image(sdA, main="shortest path length for a ring graph")
par(opar)
```

`sylvester` *Solve Sylvester Equation*

Description

The Sylvester equation is of form

$$AX + XB = C$$

where X is the unknown and others are given. Though it's possible to have non-square A and B matrices, we currently support square matrices only. This is a wrapper of `armadillo`'s `sylvester` function.

Usage

```
sylvester(A, B, C)
```

Arguments

`A`	a $(p \times p)$ matrix as above.
`B`	a $(p \times p)$ matrix as above.
`C`	a $(p \times p)$ matrix as above.

Value

a solution matrix X of size $(p \times p)$.

References

Sanderson C, Curtin R (2016). “Armadillo: A Template-Based C++ Library for Linear Algebra.” *The Journal of Open Source Software*, **1**(2), 26.

Eddelbuettel D, Sanderson C (2014). “RcppArmadillo: Accelerating R with High-Performance C++ Linear Algebra.” *Computational Statistics and Data Analysis*, **71**, 1054–1063.

Examples

```
## simulated example
#  generate square matrices
A = matrix(rnorm(25),nrow=5)
X = matrix(rnorm(25),nrow=5)
B = matrix(rnorm(25),nrow=5)
C = A%*%X + X%*%B

#  solve using 'sylvester' function
solX = sylvester(A,B,C)
pm1 = "* Experiment with Sylvester Solver"
pm2 = paste("* Absolute Error  : ",norm(solX-X,"f"),sep="")
pm3 = paste("* Relative Error  : ",norm(solX-X,"f")/norm(X,"f"),sep="")
cat(paste(pm1,"\n",pm2,"\n",pm3,sep=""))
```

`trio` *Trace Ratio Optimation*

Description

This function provides several algorithms to solve the following problem

$$\max \frac{tr(V^\top AV)}{tr(V^\top BV)} \text{ such that } V^\top CV = I$$

where V is a projection matrix, i.e., $V^\top V = I$. Trace ratio optimization is pertained to various linear dimension reduction methods. It should be noted that when $C = I$, the above problem is often reformulated as a generalized eigenvalue problem since it’s an easier proxy with faster computation.

Usage

```
trio(
  A,
  B,
  C,
  dim = 2,
  method = c("2003Guo", "2007Wang", "2009Jia", "2012Ngo"),
  maxiter = 1000,
  eps = 1e-10
)
```

Arguments

`A`	a $(p \times p)$ symmetric matrix in the numerator term.
`B`	a $(p \times p)$ symmetric matrix in the denomiator term.
`C`	a $(p \times p)$ symmetric constraint matrix. If not provided, it is set as identical matrix automatically.
`dim`	an integer for target dimension. It can be considered as the number of loadings.
`method`	the name of algorithm to be used. Default is `2003Guo`.
`maxiter`	maximum number of iterations to be performed.
`eps`	stopping criterion for iterative algorithms.

Value

a named list containing

V a $(p \times dim)$ projection matrix.

tr.val an attained maximum scalar value.

References

Guo Y, Li S, Yang J, Shu T, Wu L (2003). “A Generalized Foley–Sammon Transform Based on Generalized Fisher Discriminant Criterion and Its Application to Face Recognition.” *Pattern Recognition Letters*, **24**(1-3), 147–158.

Wang H, Yan S, Xu D, Tang X, Huang T (2007). “Trace Ratio vs. Ratio Trace for Dimensionality Reduction.” In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, 1–8.

Yangqing Jia, Feiping Nie, Changshui Zhang (2009). “Trace Ratio Problem Revisited.” *IEEE Transactions on Neural Networks*, **20**(4), 729–735.

Ngo TT, Bellalij M, Saad Y (2012). “The Trace Ratio Optimization Problem.” *SIAM Review*, **54**(3), 545–569.

Examples

```
## simple test
#  problem setting
p = 5
```

```
mydim = 2
A = matrix(rnorm(p^2),nrow=p); A=A%*%t(A)
B = matrix(runif(p^2),nrow=p); B=B%*%t(B)
C = diag(p)

#  approximate solution via determinant ratio problem formulation
eigAB  = eigen(solve(B,A))
V      = eigAB$vectors[,1:mydim]
eigval = sum(diag(t(V)%*%A%*%V))/sum(diag(t(V)%*%B%*%V))

#  solve using 4 algorithms
m12 = trio(A,B,dim=mydim, method="2012Ngo")
m09 = trio(A,B,dim=mydim, method="2009Jia")
m07 = trio(A,B,dim=mydim, method="2007Wang")
m03 = trio(A,B,dim=mydim, method="2003Guo")

#  print the results
line1 = '* Evaluation of the cost function'
line2 = paste("* approx. via determinant : ",eigval,sep="")
line3 = paste("* trio by 2012Ngo         : ",m12$tr.val, sep="")
line4 = paste("* trio by 2009Jia         : ",m09$tr.val, sep="")
line5 = paste("* trio by 2007Wang        : ",m07$tr.val, sep="")
line6 = paste("* trio by 2003Guo         : ",m03$tr.val, sep="")
cat(line1,"\n",line2,"\n",line3,"\n",line4,"\n",line5,"\n",line6)
```

tsne *t-SNE Embedding*

Description

This function is a simple wrapper of Rtsne function for t-Stochastic Neighbor Embedding for finding low-dimensional structure of the data embedded in the high-dimensional space.

Usage

```
tsne(data, ndim = 2, ...)
```

Arguments

data	an $(n \times p)$ matrix whose rows are observations.
ndim	an integer-valued target dimension.
...	extra parameters to be used in Rtsne function.

Value

a named list containing

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

stress discrepancy between embedded and origianl data as a measure of error.

Examples

```
## use simple example of iris dataset
data(iris)
mydat = as.matrix(iris[,1:4])
mylab = as.factor(iris[,5])

## run t-SNE and MDS for comparison
iris.cmds = cmds(mydat, ndim=2)
iris.tsne = tsne(mydat, ndim=2)

## extract coordinates and class information
cx = iris.cmds$embed # embedded coordinates of CMDS
tx = iris.tsne$embed #                         t-SNE

## visualize
#  main title
mc = paste("CMDS with STRESS=",round(iris.cmds$stress,4),sep="")
mt = paste("tSNE with STRESS=",round(iris.tsne$stress,4),sep="")

#  draw a figure
opar <- par(no.readonly=TRUE)
par(mfrow=c(1,2))
plot(cx, col=mylab, pch=19, main=mc)
plot(tx, col=mylab, pch=19, main=mt)
par(opar)
```

`weiszfeld` *Weiszfeld Algorithm for Computing L1-median*

Description

Geometric median, also known as L1-median, is a solution to the following problem

$$\text{argmin} \sum_{i=1}^{n} \|x_i - y\|_2$$

for a given data $x_1, x_2, \ldots, x_n \in R^p$.

Usage

```
weiszfeld(X, weights = NULL, maxiter = 496, abstol = 1e-06)
```

Arguments

`X`	an $(n \times p)$ matrix for p-dimensional signal. If vector is given, it is assumed that $p = 1$.
`weights`	`NULL` for equal weight `rep(1/n,n)`; otherwise, it has to be a vector of length n.

`maxiter`	maximum number of iterations.
`abstol`	stopping criterion

Examples

```
## generate sin(x) data with noise for 100 replicates
set.seed(496)
t = seq(from=0,to=10,length.out=20)
X = array(0,c(100,20))
for (i in 1:100){
   X[i,] = sin(t) + stats::rnorm(20, sd=0.5)
}

## compute L1-median and L2-mean
vecL2 = base::colMeans(X)
vecL1 = weiszfeld(X)

## visualize
opar <- par(no.readonly=TRUE)
par(mfrow=c(1,3), pty="s")
matplot(t(X[1:5,]), type="l", main="5 generated data", ylim=c(-2,2))
plot(t, vecL2, type="l", col="blue", main="L2-mean",   ylim=c(-2,2))
plot(t, vecL1, type="l", col="red",  main="L1-median", ylim=c(-2,2))
par(opar)
```

`WLbarycenter` *Barycenter of vMF Distributions Under a Wasserstein-Like Geometry*

Description

Given a collection of von Mises-Fisher (vMF) distributions, each characterized by a mean direction μ and a concentration parameter κ, this function solves the geometric mean problem to compute the barycentric vMF distribution under an approximate Wasserstein geometry.

Usage

```
WLbarycenter(means, concentrations, weights = NULL)
```

Arguments

`means`	An $(n \times p)$ matrix where each row represents the mean direction of one of the n vMF distributions.
`concentrations`	A length-n vector of nonnegative concentration parameters.
`weights`	A weight vector of length n. If `NULL`, equal weights (`rep(1/n, n)`) are used.

Value

A named list containing:

mean A length-p vector representing the barycenter direction.

concentration A scalar representing the barycenter concentration.

Examples

```
# Set seed for reproducibility
set.seed(123)

# Number of vMF distributions
n <- 5

# Generate mean directions concentrated around a specific angle (e.g., 45 degrees)
base_angle <- pi / 4  # 45 degrees in radians
angles <- rnorm(n, mean = base_angle, sd = pi / 20)  # Small deviation from base_angle
means <- cbind(cos(angles), sin(angles))  # Convert angles to unit vectors

# Generate concentration parameters with large magnitudes (tight distributions)
concentrations <- rnorm(n, mean = 50, sd = 5)  # Large values around 50

# Compute the barycenter under the Wasserstein-like geometry
barycenter <- WLbarycenter(means, concentrations)

# Convert barycenter mean direction to an angle
bary_angle <- atan2(barycenter$mean[2], barycenter$mean[1])

## Visualize
opar <- par(no.readonly=TRUE)
par(mfrow=c(1,2), pty="s")

# Plot the unit circle
plot(cos(seq(0, 2 * pi, length.out = 200)), sin(seq(0, 2 * pi, length.out = 200)),
     type = "l", col = "gray", lwd = 2, xlab = "x", ylab = "y",
     main = "Barycenter of vMF Distributions on S^1")

# Add input mean directions
points(means[,1], means[,2], col = "blue", pch = 19, cex = 1.5)

# Add the computed barycenter
points(cos(bary_angle), sin(bary_angle), col = "red", pch = 17, cex = 2)

# Add legend
legend("bottomleft", legend = c("vMF Means", "Barycenter"), col = c("blue", "red"),
       pch = c(19, 17), cex = 1)

# Plot the concentration parameters
hist(concentrations, main = "Concentration Parameters", xlab = "Concentration")
abline(v=barycenter$concentration, col="red", lwd=2)
par(opar)
```

WLpdist *Pairwise Wasserstein-like Distance between two vMF distributions*

Description

Given a collection of von Misees-Fisher (vMF) distributions, compute the pairwise distance using the Wasserstein-like distance from an approximate Wasserstein geometry.

Usage

```
WLpdist(means, concentrations)
```

Arguments

`means`	An $(n \times p)$ matrix where each row represents the mean direction of one of the n vMF distributions.
`concentrations`	A length-n vector of nonnegative concentration parameters.

Value

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

Examples

```
# Set seed for reproducibility
set.seed(123)

# Generate two classes of mean directions around north and south poles
means1 = array(0,c(50,2)); means1[,2] = rnorm(50, mean=1, sd=0.25)
means2 = array(0,c(50,2)); means2[,2] = rnorm(50, mean=-1, sd=0.25)
means1 = means1/sqrt(rowSums(means1^2))
means2 = means2/sqrt(rowSums(means2^2))

# Concatenate the mean directions
data_means = rbind(means1, means2)

# Generate concentration parameters
data_concentrations = rnorm(100, mean=20, sd=1)

# Compute the pairwise distance matrix
pdmat = WLpdist(data_means, data_concentrations)

# Visualise the pairwise distance matrix
opar <- par(no.readonly=TRUE)
image(pdmat, main="Pairwise Wasserstein-like Distance")
par(opar)
```

Index