Package 'T4transport'

May 7, 2026

Type Package

Title Tools for Computational Optimal Transport

Version 0.1.8

Description Transport theory has seen much success in many fields of statistics and machine learning. We provide a variety of algorithms to compute Wasserstein distance, barycenter, and others. See Peyré and Cuturi (2019) <doi:10.1561/2200000073> for the general exposition to the study of computational optimal transport.

License MIT + file LICENSE

Imports Rcpp (>= 1.1.0), Rdpack, stats, utils

LinkingTo Rcpp, RcppArmadillo

Encoding UTF-8

URL https://www.kisungyou.com/T4transport/

RoxygenNote 7.3.3

RdMacros Rdpack

Depends R (>= 2.10)

Suggests knitr, rmarkdown, ggplot2, mlbench

SystemRequirements C++20

VignetteBuilder knitr

NeedsCompilation yes

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

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

Repository CRAN

Date/Publication 2026-01-11 03:30:02 UTC

Contents

`digit3` *MNIST Images of Digit 3*

Description

`digit3` contains 2000 images from the famous MNIST dataset of digit 3. Each element of the list is an image represented as an (28×28) matrix that sums to 1. This normalization is conventional and it does not hurt its visualization via a basic 'image()' function.

Usage

```
data(digit3)
```

Format

a length-2000 named list "digit3" of (28×28) matrices.

Examples

```
## LOAD THE DATA
data(digit3)

## SHOW A FEW
opar <- par(no.readonly=TRUE)
par(mfrow=c(2,4), pty="s")
for (i in 1:8){
  image(digit3[[i]])
}
par(opar)
```

---

digits *MNIST Images of All Digits*

Description

digits contains 5000 images from the famous MNIST dataset of all digits, consisting of 500 images per digit class from 0 to 9. Each digit image is represented as an (28×28) matrix that sums to 1. This normalization is conventional and it does not hurt its visualization via a basic 'image()' function.

Usage

```
data(digits)
```

Format

a named list "digits" containing

image length-5000 list of (28×28) image matrices.

label length-5000 vector of class labels from 0 to 9.

Examples

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

## SHOW A FEW
#  Select 9 random images
subimgs = digits$image[sample(1:5000, 9)]

opar <- par(no.readonly=TRUE)
par(mfrow=c(3,3), pty="s")
for (i in 1:9){
  image(subimgs[[i]])
}
par(opar)
```

`ecdfbary` *Barycenter of Empirical CDFs*

Description

Given a collection of empirical cumulative distribution functions $F^i(x)$ for $i = 1, \ldots, N$, compute the Wasserstein barycenter of order 2. This is obtained by taking a weighted average on a set of corresponding quantile functions.

Usage

```
ecdfbary(ecdfs, weights = NULL, ...)
```

Arguments

`ecdfs`	a length-N list of `"ecdf"` objects by [stats::ecdf()].
`weights`	a weight of each image; if `NULL` (default), uniform weight is set. Otherwise, it should be a length-N vector of nonnegative weights.
`...`	extra parameters including
	abstol stopping criterion for iterations (default: 1e-8).
	maxiter maximum number of iterations (default: 496).

Value

an `"ecdf"` object of the Wasserstein barycenter.

Examples

```
#-----------------------------------------------------------------------
#                          Two Gaussians
#
# Two Gaussian distributions are parametrized as follows.
# Type 1 : (mean, var) = (-4, 1/4)
# Type 2 : (mean, var) = (+4, 1/4)
#-----------------------------------------------------------------------
# GENERATE ECDFs
ecdf_list = list()
ecdf_list[[1]] = stats::ecdf(stats::rnorm(200, mean=-4, sd=0.5))
ecdf_list[[2]] = stats::ecdf(stats::rnorm(200, mean=+4, sd=0.5))

# COMPUTE THE BARYCENTER OF EQUAL WEIGHTS
emean = ecdfbary(ecdf_list)

# QUANTITIES FOR PLOTTING
x_grid  = seq(from=-8, to=8, length.out=100)
y_type1 = ecdf_list[[1]](x_grid)
y_type2 = ecdf_list[[2]](x_grid)
y_bary  = emean(x_grid)

# VISUALIZE
opar <- par(no.readonly=TRUE)
plot(x_grid, y_bary, lwd=3, col="red", type="l",
     main="Barycenter", xlab="x", ylab="Fn(x)")
lines(x_grid, y_type1, col="gray50", lty=3)
lines(x_grid, y_type2, col="gray50", lty=3)
par(opar)
```

`ecdfmed` *Wasserstein Median of Empirical CDFs*

Description

Given a collection of empirical cumulative distribution functions $F^i(x)$ for $i = 1, \ldots, N$, compute the Wasserstein median. This is obtained by a functional variant of the Weiszfeld algorithm on a set of quantile functions.

Usage

```
ecdfmed(ecdfs, weights = NULL, ...)
```

Arguments

`ecdfs`	a length-N list of `"ecdf"` objects by [stats::ecdf()].
`weights`	a weight of each image; if `NULL` (default), uniform weight is set. Otherwise, it should be a length-N vector of nonnegative weights.

... extra parameters including

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

maxiter maximum number of iterations (default: 496).

Value

an "ecdf" object of the Wasserstein median.

Examples

```
#-----------------------------------------------------------------------
#                         Tree Gaussians
#
# Three Gaussian distributions are parametrized as follows.
# Type 1 : (mean, sd) = (-4, 1)
# Type 2 : (mean, sd) = ( 0, 1/5)
# Type 3 : (mean, sd) = (+6, 1/2)
#-----------------------------------------------------------------------
# GENERATE ECDFs
ecdf_list = list()
ecdf_list[[1]] = stats::ecdf(stats::rnorm(200, mean=-4, sd=1))
ecdf_list[[2]] = stats::ecdf(stats::rnorm(200, mean=+4, sd=0.2))
ecdf_list[[3]] = stats::ecdf(stats::rnorm(200, mean=+6, sd=0.5))

# COMPUTE THE MEDIAN
emeds = ecdfmed(ecdf_list)

# COMPUTE THE BARYCENTER
emean = ecdfbary(ecdf_list)

# QUANTITIES FOR PLOTTING
x_grid  = seq(from=-8, to=10, length.out=500)
y_type1 = ecdf_list[[1]](x_grid)
y_type2 = ecdf_list[[2]](x_grid)
y_type3 = ecdf_list[[3]](x_grid)

y_bary = emean(x_grid)
y_meds = emeds(x_grid)

# VISUALIZE
opar <- par(no.readonly=TRUE)
plot(x_grid, y_bary, lwd=3, col="orange", type="l",
     main="Wasserstein Median & Barycenter",
     xlab="x", ylab="Fn(x)", lty=2)
lines(x_grid, y_meds, lwd=3, col="blue", lty=2)
lines(x_grid, y_type1, col="gray50", lty=3)
lines(x_grid, y_type2, col="gray50", lty=3)
lines(x_grid, y_type3, col="gray50", lty=3)
legend("topleft", legend=c("Median","Barycenter"),
        lwd=3, lty=2, col=c("blue","orange"))
par(opar)
```

`fbary14C` *Fixed-Support Barycenter by Cuturi & Doucet (2014)*

Description

Given K empirical measures $\mu_1, \mu_2, \ldots, \mu_K$ of possibly different cardinalities, wasserstein barycenter μ^* is the solution to the following problem

$$\sum_{k=1}^{K} \pi_k \mathcal{W}_p^p(\mu, \mu_k)$$

where π_k's are relative weights of empirical measures. Here we assume either (1) support atoms in Euclidean space are given, or (2) all pairwise distances between atoms of the fixed support and empirical measures are given. Algorithmically, it is a subgradient method where the each subgradient is approximated using the entropic regularization.

Usage

```
fbary14C(
  support,
  atoms,
  marginals = NULL,
  weights = NULL,
  lambda = 0.1,
  p = 2,
  ...
)

fbary14Cdist(
  distances,
  marginals = NULL,
  weights = NULL,
  lambda = 0.1,
  p = 2,
  ...
)
```

Arguments

`support`	an $(N \times P)$ matrix of rows being atoms for the fixed support.
`atoms`	a length-K list where each element is an $(N_k \times P)$ matrix of atoms.
`marginals`	marginal distribution for empirical measures; if `NULL` (default), uniform weights are set for all measures. Otherwise, it should be a length-K list where each element is a length-N_i vector of nonnegative weights that sum to 1.

weights	weights for each individual measure; if NULL (default), each measure is considered equally. Otherwise, it should be a length-K vector.
lambda	regularization parameter (default: 0.1).
p	an exponent for the order of the distance (default: 2).
...	extra parameters including
	abstol stopping criterion for iterations (default: 1e-10).
	init.vec an initial vector (default: uniform weight).
	maxiter maximum number of iterations (default: 496).
	print.progress a logical to show current iteration (default: FALSE).
distances	a length-K list where each element is an $(N \times N_k)$ pairwise distance between atoms of the fixed support and given measures.

Value

a length-N vector of probability vector.

References

Cuturi M, Doucet A (2014-06-22/2014-06-24). "Fast Computation of Wasserstein Barycenters." In Xing EP, Jebara T (eds.), *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, 685–693.

Examples

```
#-----------------------------------------------------------------
#     Wasserstein Barycenter for Fixed Atoms with Two Gaussians
#
# * class 1 : samples from Gaussian with mean=(-4, -4)
# * class 2 : samples from Gaussian with mean=(+4, +4)
# * target support consists of 7 integer points from -6 to 6,
#   where ideally, weight is concentrated near 0 since it's average!
#-----------------------------------------------------------------
## GENERATE DATA
#  Empirical Measures
set.seed(100)
ndat = 100
dat1 = matrix(rnorm(ndat*2, mean=-4, sd=0.5),ncol=2)
dat2 = matrix(rnorm(ndat*2, mean=+4, sd=0.5),ncol=2)

myatoms = list()
myatoms[[1]] = dat1
myatoms[[2]] = dat2
mydata = rbind(dat1, dat2)

#  Fixed Support
support = cbind(seq(from=-8,to=8,by=2),
                seq(from=-8,to=8,by=2))
## COMPUTE
comp1 = fbary14C(support, myatoms, lambda=0.5, maxiter=10)
```

```
comp2 = fbary14C(support, myatoms, lambda=1,   maxiter=10)
comp3 = fbary14C(support, myatoms, lambda=10,  maxiter=10)

## VISUALIZE
opar <- par(no.readonly=TRUE)
par(mfrow=c(1,3), pty="s")
barplot(comp1, ylim=c(0,1), main="Probability\n (lambda=0.5)")
barplot(comp2, ylim=c(0,1), main="Probability\n (lambda=1)")
barplot(comp3, ylim=c(0,1), main="Probability\n (lambda=10)")
par(opar)
```

`fbary15B` *Fixed-Support Barycenter by Benamou et al. (2015)*

Description

Given K empirical measures $\mu_1, \mu_2, \ldots, \mu_K$ of possibly different cardinalities, wasserstein barycenter μ^* is the solution to the following problem

$$\sum_{k=1}^{K} \pi_k \mathcal{W}_p^p(\mu, \mu_k)$$

where π_k's are relative weights of empirical measures. Here we assume either (1) support atoms in Euclidean space are given, or (2) all pairwise distances between atoms of the fixed support and empirical measures are given. Authors proposed iterative Bregman projections in conjunction with entropic regularization.

Usage

```
fbary15B(
  support,
  atoms,
  marginals = NULL,
  weights = NULL,
  lambda = 0.1,
  p = 2,
  ...
)

fbary15Bdist(
  distances,
  marginals = NULL,
  weights = NULL,
  lambda = 0.1,
  p = 2,
  ...
)
```

Arguments

support	an $(N \times P)$ matrix of rows being atoms for the fixed support.
atoms	a length-K list where each element is an $(N_k \times P)$ matrix of atoms.
marginals	marginal distribution for empirical measures; if NULL (default), uniform weights are set for all measures. Otherwise, it should be a length-K list where each element is a length-N_i vector of nonnegative weights that sum to 1.
weights	weights for each individual measure; if NULL (default), each measure is considered equally. Otherwise, it should be a length-K vector.
lambda	regularization parameter (default: 0.1).
p	an exponent for the order of the distance (default: 2).
...	extra parameters including
	abstol stopping criterion for iterations (default: 1e-10).
	init.vec an initial vector (default: uniform weight).
	maxiter maximum number of iterations (default: 496).
	print.progress a logical to show current iteration (default: FALSE).
distances	a length-K list where each element is an $(N \times N_k)$ pairwise distance between atoms of the fixed support and given measures.

Value

a length-N vector of probability vector.

References

Benamou J, Carlier G, Cuturi M, Nenna L, Peyré G (2015). "Iterative Bregman Projections for Regularized Transportation Problems." *SIAM Journal on Scientific Computing*, **37**(2), A1111-A1138. ISSN 1064-8275, 1095-7197, doi:10.1137/141000439.

Examples

```
#-----------------------------------------------------------------
#     Wasserstein Barycenter for Fixed Atoms with Two Gaussians
#
# * class 1 : samples from Gaussian with mean=(-4, -4)
# * class 2 : samples from Gaussian with mean=(+4, +4)
# * target support consists of 7 integer points from -6 to 6,
#   where ideally, weight is concentrated near 0 since it's average!
#-----------------------------------------------------------------
## GENERATE DATA
#  Empirical Measures
set.seed(100)
ndat = 500
dat1 = matrix(rnorm(ndat*2, mean=-4, sd=0.5),ncol=2)
dat2 = matrix(rnorm(ndat*2, mean=+4, sd=0.5),ncol=2)

myatoms = list()
myatoms[[1]] = dat1
```

```
myatoms[[2]] = dat2
mydata = rbind(dat1, dat2)

#  Fixed Support
support = cbind(seq(from=-8,to=8,by=2),
                seq(from=-8,to=8,by=2))
## COMPUTE
comp1 = fbary15B(support, myatoms, lambda=0.5, maxiter=10)
comp2 = fbary15B(support, myatoms, lambda=1,   maxiter=10)
comp3 = fbary15B(support, myatoms, lambda=10,  maxiter=10)

## VISUALIZE
opar <- par(no.readonly=TRUE)
par(mfrow=c(1,3), pty="s")
barplot(comp1, ylim=c(0,1), main="Probability\n (lambda=0.5)")
barplot(comp2, ylim=c(0,1), main="Probability\n (lambda=1)")
barplot(comp3, ylim=c(0,1), main="Probability\n (lambda=10)")
par(opar)
```

`fiedler` *Compute the fiedler vector of a point cloud*

Description

Given a point cloud $X \in \mathbf{R}^{N \times P}$, this function constructs a fully connected weighted graph using an RBF (Gaussian) kernel with bandwidth chosen by the median heuristic, forms the unnormalized graph Laplacian, and returns the corresponding Fiedler vector, which is the eigenvector associated to the second smallest eigenvalue of the Laplacian.

Usage

```
fiedler(X, normalize = TRUE)
```

Arguments

`X`	An $(N \times P)$ matrix of row observations.
`normalize`	Logical; if `TRUE` (default), the Fiedler vector is rescaled to lie in $[0, 1]$ by subtracting its minimum and dividing by its range, mimicking the normalization convention in the corresponding Python implementation. If `FALSE`, the raw eigenvector is returned.

Value

A numeric vector of length N containing the Fiedler values associated with each point in the input point cloud. If `normalize = TRUE`, the entries are in the interval $[0, 1]$.

Examples

```
#---------------------------------------------------------------------
#                           Description
#
# Use 'iris' dataset to compute fiedler vector.
# The dataset is visualized in R^2 using PCA
#---------------------------------------------------------------------
# load dataset
X = as.matrix(iris[,1:4])

# PCA preprocessing
X2d = X%*%eigen(cov(X))$vectors[,1:2]

# compute fiedler vector
fied_vec = fiedler(X2d, normalize=TRUE)

# plot
opar <- par(no.readonly=TRUE)
plot(X2d, col=rainbow(150)[as.numeric(cut(fied_vec, breaks=150))],
     pch=19, xlab="PC 1", ylab="PC 2",
     main="Fiedler vector on Iris dataset (PCA-reduced)")
par(opar)
```

---

gaussbary1d *Barycenter of Gaussian Distributions in* $\mathbb{R}$

Description

Given a collection of Gaussian distributions $\mathcal{N}(\mu_i, \sigma_i^2)$ for $i = 1, \ldots, n$, compute the Wasserstein barycenter of order 2. For the barycenter computation of variance components, we use a fixed-point algorithm by Álvarez-Esteban et al. (2016).

Usage

```
gaussbary1d(means, vars, weights = NULL, ...)
```

Arguments

means	a length-n vector of mean parameters.
vars	a length-n vector of variance parameters.
weights	a weight of each image; if NULL (default), uniform weight is set. Otherwise, it should be a length-n vector of nonnegative weights.
...	extra parameters including **abstol** stopping criterion for iterations (default: 1e-8). **maxiter** maximum number of iterations (default: 496).

Value

a named list containing

mean mean of the estimated barycenter distribution.

var variance of the estimated barycenter distribution.

References

Álvarez-Esteban PC, del Barrio E, Cuesta-Albertos JA, Matrán C (2016). "A Fixed-Point Approach to Barycenters in Wasserstein Space." *Journal of Mathematical Analysis and Applications*, **441**(2), 744–762. ISSN 0022247X, doi:10.1016/j.jmaa.2016.04.045.

See Also

[T4transport::gaussbarypd()] for multivariate case.

Examples

```
#-----------------------------------------------------------------------
#                          Two Gaussians
#
# Two Gaussian distributions are parametrized as follows.
# Type 1 : (mean, var) = (-4, 1/4)
# Type 2 : (mean, var) = (+4, 1/4)
#-----------------------------------------------------------------------
# GENERATE PARAMETERS
par_mean = c(-4, 4)
par_vars = c(0.25, 0.25)

# COMPUTE THE BARYCENTER OF EQUAL WEIGHTS
gmean = gaussbary1d(par_mean, par_vars)

# QUANTITIES FOR PLOTTING
x_grid  = seq(from=-6, to=6, length.out=200)
y_dist1 = stats::dnorm(x_grid, mean=-4, sd=0.5)
y_dist2 = stats::dnorm(x_grid, mean=+4, sd=0.5)
y_gmean = stats::dnorm(x_grid, mean=gmean$mean, sd=sqrt(gmean$var))

# VISUALIZE
opar <- par(no.readonly=TRUE)
plot(x_grid, y_gmean, lwd=2, col="red", type="l",
     main="Barycenter", xlab="x", ylab="density")
lines(x_grid, y_dist1)
lines(x_grid, y_dist2)
par(opar)
```

gaussbarypd *Barycenter of Gaussian Distributions in $\mathbb{R}\hat{}p$*

Description

Given a collection of n-dimensional Gaussian distributions $N(\mu_i, \Sigma_i)$ for $i = 1, \ldots, n$, compute the Wasserstein barycenter of order 2. For the barycenter computation of variance components, we use a fixed-point algorithm by Álvarez-Esteban et al. (2016).

Usage

```
gaussbarypd(means, vars, weights = NULL, ...)
```

Arguments

`means`	an $(n \times p)$ matrix whose rows are mean vectors.
`vars`	a $(p \times p \times n)$ array where each slice is covariance matrix.
`weights`	a weight of each image; if `NULL` (default), uniform weight is set. Otherwise, it should be a length-n vector of nonnegative weights.
`...`	extra parameters including **abstol** stopping criterion for iterations (default: 1e-8). **maxiter** maximum number of iterations (default: 496).

Value

a named list containing

mean a length-p vector for mean of the estimated barycenter distribution.

var a $(p \times p)$ matrix for variance of the estimated barycenter distribution.

References

Álvarez-Esteban PC, del Barrio E, Cuesta-Albertos JA, Matrán C (2016). "A Fixed-Point Approach to Barycenters in Wasserstein Space." *Journal of Mathematical Analysis and Applications*, **441**(2), 744–762. ISSN 0022247X, doi:10.1016/j.jmaa.2016.04.045.

See Also

[T4transport::gaussbary1d()] for univariate case.

Examples

```
#-----------------------------------------------------------------------
#                          Two Gaussians in R^2
#-----------------------------------------------------------------------
# GENERATE PARAMETERS
# means
par_mean = rbind(c(-4,0), c(4,0))

# covariances
par_vars = array(0,c(2,2,2))
par_vars[,,1] = cbind(c(4,-2),c(-2,4))
par_vars[,,2] = cbind(c(4,+2),c(+2,4))

# COMPUTE THE BARYCENTER OF EQUAL WEIGHTS
gmean = gaussbarypd(par_mean, par_vars)

# GET COORDINATES FOR DRAWING
pt_type1 = gaussvis2d(par_mean[1,], par_vars[,,1])
pt_type2 = gaussvis2d(par_mean[2,], par_vars[,,2])
pt_gmean = gaussvis2d(gmean$mean, gmean$var)

# VISUALIZE
opar <- par(no.readonly=TRUE)
plot(pt_gmean, lwd=2, col="red", type="l",
     main="Barycenter", xlab="", ylab="",
     xlim=c(-6,6))
lines(pt_type1)
lines(pt_type2)
par(opar)
```

`gaussmed1d` *Wasserstein Median of Gaussian Distributions in* $\mathbb{R}$

Description

Given a collection of Gaussian distributions $\mathcal{N}(\mu_i, \sigma_i^2)$ for $i = 1, \ldots, n$, compute the Wasserstein median.

Usage

```
gaussmed1d(means, vars, weights = NULL, ...)
```

Arguments

`means`	a length-n vector of mean parameters.
`vars`	a length-n vector of variance parameters.

`weights`	a weight of each image; if `NULL` (default), uniform weight is set. Otherwise, it should be a length-n vector of nonnegative weights.
`...`	extra parameters including **abstol** stopping criterion for iterations (default: 1e-8). **maxiter** maximum number of iterations (default: 496).

Value

a named list containing

mean mean of the estimated median distribution.

var variance of the estimated median distribution.

References

You K, Shung D, Giuffrè M (2025). "On the Wasserstein Median of Probability Measures." *Journal of Computational and Graphical Statistics*, **34**(1), 253-266. ISSN 1061-8600, 1537-2715.

See Also

[T4transport::gaussmedpd()] for multivariate case.

Examples

```
#-----------------------------------------------------------------------
#                         Tree Gaussians
#
# Three Gaussian distributions are parametrized as follows.
# Type 1 : (mean, sd) = (-4, 1)
# Type 2 : (mean, sd) = ( 0, 1/5)
# Type 3 : (mean, sd) = (+6, 1/2)
#-----------------------------------------------------------------------
# GENERATE PARAMETERS
par_mean = c(-4, 0, +6)
par_vars = c(1, 0.04, 0.25)

# COMPUTE THE WASSERSTEIN MEDIAN
gmeds = gaussmed1d(par_mean, par_vars)

# COMPUTE THE BARYCENTER
gmean = gaussbary1d(par_mean, par_vars)

# QUANTITIES FOR PLOTTING
x_grid  = seq(from=-6, to=8, length.out=1000)
y_dist1 = stats::dnorm(x_grid, mean=par_mean[1], sd=sqrt(par_vars[1]))
y_dist2 = stats::dnorm(x_grid, mean=par_mean[2], sd=sqrt(par_vars[2]))
y_dist3 = stats::dnorm(x_grid, mean=par_mean[3], sd=sqrt(par_vars[3]))

y_gmean = stats::dnorm(x_grid, mean=gmean$mean, sd=sqrt(gmean$var))
y_gmeds = stats::dnorm(x_grid, mean=gmeds$mean, sd=sqrt(gmeds$var))
```

```
# VISUALIZE
opar <- par(no.readonly=TRUE)
plot(x_grid, y_gmeds, lwd=3, col="red", type="l",
     main="Three Gaussians", xlab="x", ylab="density",
     xlim=range(x_grid), ylim=c(0,2.5))
lines(x_grid, y_gmean, lwd=3, col="blue")
lines(x_grid, y_dist1, lwd=1.5, lty=2)
lines(x_grid, y_dist2, lwd=1.5, lty=2)
lines(x_grid, y_dist3, lwd=1.5, lty=2)
legend("topleft", legend=c("Median","Barycenter"),
       col=c("red","blue"), lwd=c(3,3), lty=c(1,2))
par(opar)
```

`gaussmedpd` *Wasserstein Median of Gaussian Distributions in* $\mathbb{R}\hat{}p$

Description

Given a collection of p-dimensional Gaussian distributions $N(\mu_i, \Sigma_i)$ for $i = 1, \ldots, n$, compute the Wasserstein median.

Usage

```
gaussmedpd(means, vars, weights = NULL, ...)
```

Arguments

`means`	an $(n \times p)$ matrix whose rows are mean vectors.
`vars`	a $(p \times p \times n)$ array where each slice is covariance matrix.
`weights`	a weight of each image; if `NULL` (default), uniform weight is set. Otherwise, it should be a length-n vector of nonnegative weights.
`...`	extra parameters including **abstol** stopping criterion for iterations (default: 1e-8). **maxiter** maximum number of iterations (default: 496).

Value

a named list containing

mean a length-p vector for mean of the estimated median distribution.

var a $(p \times p)$ matrix for variance of the estimated median distribution.

References

You K, Shung D, Giuffrè M (2025). "On the Wasserstein Median of Probability Measures." *Journal of Computational and Graphical Statistics*, **34**(1), 253-266. ISSN 1061-8600, 1537-2715.

See Also

[T4transport::gaussmed1d()] for univariate case.

Examples

```
#-----------------------------------------------------------------------
#                          Three Gaussians in R^2
#-----------------------------------------------------------------------
# GENERATE PARAMETERS
# means
par_mean = rbind(c(-4,0), c(0,0), c(5,-1))

# covariances
par_vars = array(0,c(2,2,3))
par_vars[,,1] = cbind(c(2,-1),c(-1,2))
par_vars[,,2] = cbind(c(4,+1),c(+1,4))
par_vars[,,3] = diag(c(4,1))

# COMPUTE THE MEDIAN
gmeds = gaussmedpd(par_mean, par_vars)

# COMPUTE THE BARYCENTER
gmean = gaussbarypd(par_mean, par_vars)

# GET COORDINATES FOR DRAWING
pt_type1 = gaussvis2d(par_mean[1,], par_vars[,,1])
pt_type2 = gaussvis2d(par_mean[2,], par_vars[,,2])
pt_type3 = gaussvis2d(par_mean[3,], par_vars[,,3])
pt_gmean = gaussvis2d(gmean$mean, gmean$var)
pt_gmeds = gaussvis2d(gmeds$mean, gmeds$var)

# VISUALIZE
opar <- par(no.readonly=TRUE)
plot(pt_gmean, lwd=2, col="red", type="l",
     main="Three Gaussians", xlab="", ylab="",
     xlim=c(-6,8), ylim=c(-2.5,2.5))
lines(pt_gmeds, lwd=2, col="blue")
lines(pt_type1, lty=2, lwd=5)
lines(pt_type2, lty=2, lwd=5)
lines(pt_type3, lty=2, lwd=5)
abline(h=0, col="grey80", lty=3)
abline(v=0, col="grey80", lty=3)
legend("topright", legend=c("Median","Barycenter"),
       lwd=2, lty=1, col=c("blue","red"))
par(opar)
```

---

gaussvis2d *Sampling from a Bivariate Gaussian Distribution for Visualization*

Description

This function samples points along the contour of an ellipse represented by mean and variance parameters for a 2-dimensional Gaussian distribution to help ease manipulating visualization of the specified distribution. For example, you can directly use a basic `plot()` function directly for drawing.

Usage

```
gaussvis2d(mean, var, n = 500)
```

Arguments

`mean`	a length-2 vector for mean parameter.
`var`	a (2×2) matrix for covariance parameter.
`n`	the number of points to be drawn (default: 500).

Value

an $(n \times 2)$ matrix.

Examples

```
#-----------------------------------------------------------------------
#                        Three Gaussians in R^2
#-----------------------------------------------------------------------
# MEAN PARAMETERS
loc1 = c(-3,0)
loc2 = c(0,5)
loc3 = c(3,0)

# COVARIANCE PARAMETERS
var1 = cbind(c(4,-2),c(-2,4))
var2 = diag(c(9,1))
var3 = cbind(c(4,2),c(2,4))

# GENERATE POINTS
visA = gaussvis2d(loc1, var1)
visB = gaussvis2d(loc2, var2)
visC = gaussvis2d(loc3, var3)

# VISUALIZE
opar <- par(no.readonly=TRUE)
plot(visA[,1], visA[,2], type="l", xlim=c(-5,5), ylim=c(-2,9),
     lwd=3, col="red", main="3 Gaussian Distributions")
lines(visB[,1], visB[,2], lwd=3, col="blue")
lines(visC[,1], visC[,2], lwd=3, col="orange")
legend("top", legend=c("Type 1","Type 2","Type 3"),
       lwd=3, col=c("red","blue","orange"), horiz=TRUE)
par(opar)
```

gwbary *Gromov-Wasserstein Barycenter*

Description

Computes the Gromov–Wasserstein (GW) barycenter of a collection of metric measure spaces. Given a list of distance matrices $D^{(k)}k = 1^K$ and their corresponding marginal distributions, the function estimates a synthetic metric space whose intrinsic geometry best represents the input collection under the GW criterion.

The GW barycenter is defined as the minimizer of a multi-measure Gromov–Wasserstein objective, where each dataset contributes according to a user-specified barycentric weight. Since the problem is jointly non-convex in the barycenter metric and the coupling matrices, the algorithm proceeds through an outer–inner iterative procedure.

Usage

```
gwbary(distances, marginals = NULL, weights = NULL, num_support = 100, ...)
```

Arguments

`distances`	a length-K list where each element is either an $(N_k \times N_k)$ distance matrix or an object of class `dist` representing the pairwise distances for each empirical measure.
`marginals`	marginal distributions for empirical measures; if `NULL` (default), uniform weights are set for all measures. Otherwise, it should be a length-K list where each element is a length-N_k vector of nonnegative weights that sum to 1.
`weights`	weights for each individual measure; if `NULL` (default), each measure is considered equally. Otherwise, it should be a length-K vector.
`num_support`	the number of support points M for the barycenter (default: 100).
`...`	extra parameters including **maxiter** maximum number of iterations (default: 10). **abstol** stopping criterion for iterations (default: 1e-6). **method** optimization method to use; can be one of `"mm"`, `"pg"`, or `"fw"` (default).

Value

A named list containing

dist an object of class `dist` representing the GW barycenter.

weight a length-M vector of barycenter weights with all entries being $1/M$.

Examples

```
## Not run:
#-----------------------------------------------------------------
#                          Description
#
# GW barycenter computation is quite expensive. In this example,
# we draw a small set of empirical measures from the digit '3'
# images and compute their GW barycenter with a small number of
# support points. The attained barycenter distance matrix is then
# passed onto the classical MDS algorithm for visualization.
#-----------------------------------------------------------------
## GENERATE DATA
data(digits)
data_D = vector("list", length=5)
data_W = vector("list", length=5)
for (i in 1:5){
  img_now = img2measure(digits3[[i]])
  data_D[[i]] = stats::dist(img_now$support)
  data_W[[i]] = as.vector(img_now$weight)
}

## COMPUTE
bary_dist <- gwbary(data_D, marginals=data_W, num_support=100)
bary_cmd2 <- stats::cmdscale(bary_dist$dist, k=2)

## VISUALIZE
opar <- par(no.readonly=TRUE)
par(pty="s")
plot(bary_cmd2, main="GW Barycenter Embedding",
     xaxt="n", yaxt="n", pch=19, xlab="", ylab="")
par(opar)

## End(Not run)
```

---

gwdist *Gromov-Wasserstein Distance*

Description

Computes the Gromov-Wasserstein (GW) distance between two metric measure spaces. Given two distance matrices D_X and D_Y along with their respective marginal distributions, the function solves the GW optimization problem to obtain both the distance value and an associated optimal transport plan.

The GW distance provides a way to compare datasets that may not lie in the same ambient space by focusing on the intrinsic geometric structure encoded in the pairwise distances. This implementation supports multiple optimization schemes, including majorization–minimization (MM), proximal gradient (PG), and Frank–Wolfe (FW).

Usage

```
gwdist(Dx, Dy, wx = NULL, wy = NULL, ...)
```

Arguments

Dx	an $(M \times M)$ distance matrix or a `dist` object of compatible size.
Dy	an $(N \times N)$ distance matrix or a `dist` object of compatible size.
wx	a length-M marginal density that sums to 1. If `NULL` (default), uniform weight is set.
wy	a length-N marginal density that sums to 1. If `NULL` (default), uniform weight is set.
...	extra parameters including **maxiter** maximum number of iterations (default: 10). **abstol** stopping criterion for iterations (default: 1e-6). **method** optimization method to use; can be one of `"mm"`, `"pg"`, or `"fw"` (default).

Value

a named list containing

distance the computed GW distance value.

plan an $(M \times N)$ nonnegative matrix for the optimal transport plan.

References

Mémoli F (2011). "Gromov–Wasserstein Distances and the Metric Approach to Object Matching." *Foundations of Computational Mathematics*, **11**(4), 417–487. ISSN 1615-3375, 1615-3383.

Examples

```
## Not run:
#-----------------------------------------------------------------
#                           Description
#
# * class 1 : iris dataset (columns 1-4) with perturbations
# * class 2 : class 1 rotated randomly in R^4
# * class 3 : samples from N((0,0), I)
#
#  We draw 10 empirical measures from each and compare
#  the regular Wasserstein and GW distance. It is expected that
#  the GW distance between class 1 and class 2 is negligible,
#  while the regular Wasserstein distance is large. For simplicity,
#  limit the cardinalities to 20.
#-----------------------------------------------------------------
## GENERATE DATA
set.seed(10)
```

```
#  prepare empty lists
inputs = vector("list", length=30)

#  generate class 1 and 2
iris_mat = as.matrix(iris[sample(1:150,20),1:4])
for (i in 1:10){
  inputs[[i]] = iris_mat + matrix(rnorm(20*4), ncol=4)
  inputs[[i+10]] = inputs[[i]]%*%qr.Q(qr(matrix(runif(16), ncol=4)))
}
#  generate class 3
for (j in 21:30){
  inputs[[j]] = matrix(rnorm(20*4), ncol=4)
}

## COMPUTE
#  empty arrays
dist_RW = array(0, c(30, 30))
dist_GW = array(0, c(30, 30))

#  compute pairwise distances
for (i in 1:29){
  X <- inputs[[i]]
  Dx <- stats::dist(X)
  for (j in (i+1):30){
  Y <- inputs[[j]]
  Dy <- stats::dist(Y)
  dist_RW[i,j] <- dist_RW[j,i] <- wasserstein(X, Y)$distance
  dist_GW[i,j] <- dist_GW[j,i] <- gwdist(Dx, Dy)$distance
  }
}

## VISUALIZE
opar <- par(no.readonly=TRUE)
par(mfrow=c(1,2), pty="s")
image(dist_RW, xaxt="n", yaxt="n", main="Regular Wasserstein distance")
image(dist_GW, xaxt="n", yaxt="n", main="Gromov-Wasserstein distance")
par(opar)

## End(Not run)
```

`histbary` *Barycenter of Histograms*

Description

Given multiple histograms represented as `"histogram"` S3 objects, compute their 2-Wasserstein barycenter using the exact 1D quantile characterization. All input histograms must have identical breaks.

Usage

```
histbary(hists, weights = NULL, L = 2000L)
```

Arguments

hists	a length-N list of histograms ("histogram" objects) of same breaks.
weights	a weight for each histogram; if NULL (default), uniform weights are used. Otherwise, it should be a length-N vector of nonnegative weights.
L	number of quantile levels used to approximate the barycenter (default: 2000). Larger L gives a more accurate approximation at increased computational cost.

Value

a "histogram" object representing the Wasserstein barycenter.

Examples

```
#-----------------------------------------------------------------------
#                     Binned from Two Gaussians
#
# EXAMPLE : Very Small Example for CRAN; just showing how to use it!
#-----------------------------------------------------------------------
# GENERATE FROM TWO GAUSSIANS WITH DIFFERENT MEANS
set.seed(100)
x  = stats::rnorm(1000, mean=-4, sd=0.5)
y  = stats::rnorm(1000, mean=+4, sd=0.5)
bk = seq(from=-10, to=10, length.out=20)

# HISTOGRAMS WITH COMMON BREAKS
histxy = list()
histxy[[1]] = hist(x, breaks=bk, plot=FALSE)
histxy[[2]] = hist(y, breaks=bk, plot=FALSE)

# COMPUTE
hh = histbary(histxy)

# VISUALIZE
opar <- par(no.readonly=TRUE)
par(mfrow=c(1,2), pty="s")
barplot(histxy[[1]]$density, col=rgb(0,0,1,1/4),
        ylim=c(0, 0.75), main="Two Histograms")
barplot(histxy[[2]]$density, col=rgb(1,0,0,1/4),
        ylim=c(0, 0.75), add=TRUE)
barplot(hh$density, main="Barycenter",
        ylim=c(0, 0.75))
par(opar)
```

`histbary14C` *Barycenter of Histograms by Cuturi and Doucet (2014)*

Description

Given multiple histograms represented as `"histogram"` S3 objects, compute Wasserstein barycenter. We need one requirement that all histograms in an input list `hists` must have **same breaks**. See the example on how to construct a histogram on predefined breaks/bins.

Usage

```
histbary14C(hists, p = 2, weights = NULL, lambda = NULL, ...)
```

Arguments

`hists`	a length-N list of histograms (`"histogram"` object) of same breaks.
`p`	an exponent for the order of the distance (default: 2).
`weights`	a weight of each image; if `NULL` (default), uniform weight is set. Otherwise, it should be a length-N vector of nonnegative weights.
`lambda`	a regularization parameter; if `NULL` (default), a paper's suggestion would be taken, or it should be a nonnegative real number.
`...`	extra parameters including
	abstol stopping criterion for iterations (default: 1e-8).
	init.vec an initial weight vector (default: uniform weight).
	maxiter maximum number of iterations (default: 496).
	nthread number of threads for OpenMP run (default: 1).
	print.progress a logical to show current iteration (default: `TRUE`).

Value

a `"histogram"` object representing the Wasserstein barycenter.

References

Cuturi M, Doucet A (2014-06-22/2014-06-24). "Fast Computation of Wasserstein Barycenters." In Xing EP, Jebara T (eds.), *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, 685–693.

See Also

`fbary14C`

Examples

```
#---------------------------------------------------------------------
#                      Binned from Two Gaussians
#
# EXAMPLE : Very Small Example for CRAN; just showing how to use it!
#---------------------------------------------------------------------
# GENERATE FROM TWO GAUSSIANS WITH DIFFERENT MEANS
set.seed(100)
x  = stats::rnorm(1000, mean=-4, sd=0.5)
y  = stats::rnorm(1000, mean=+4, sd=0.5)
bk = seq(from=-10, to=10, length.out=20)

# HISTOGRAMS WITH COMMON BREAKS
histxy = list()
histxy[[1]] = hist(x, breaks=bk, plot=FALSE)
histxy[[2]] = hist(y, breaks=bk, plot=FALSE)

# COMPUTE
hh = histbary14C(histxy, maxiter=5)

# VISUALIZE
opar <- par(no.readonly=TRUE)
par(mfrow=c(1,2))
barplot(histxy[[1]]$density, col=rgb(0,0,1,1/4),
        ylim=c(0, 0.75), main="Two Histograms")
barplot(histxy[[2]]$density, col=rgb(1,0,0,1/4),
        ylim=c(0, 0.75), add=TRUE)
barplot(hh$density, main="Barycenter",
        ylim=c(0, 0.75))
par(opar)
```

`histbary15B` *Barycenter of Histograms by Benamou et al. (2015)*

Description

Given multiple histograms represented as `"histogram"` S3 objects, compute Wasserstein barycenter. We need one requirement that all histograms in an input list `hists` must have **same breaks**. See the example on how to construct a histogram on predefined breaks/bins.

Usage

```
histbary15B(hists, p = 2, weights = NULL, lambda = NULL, ...)
```

Arguments

`hists`	a length-N list of histograms (`"histogram"` object) of same breaks.
`p`	an exponent for the order of the distance (default: 2).
`weights`	a weight of each image; if `NULL` (default), uniform weight is set. Otherwise, it should be a length-N vector of nonnegative weights.
`lambda`	a regularization parameter; if `NULL` (default), a paper's suggestion would be taken, or it should be a nonnegative real number.
`...`	extra parameters including
	abstol stopping criterion for iterations (default: 1e-8).
	init.vec an initial weight vector (default: uniform weight).
	maxiter maximum number of iterations (default: 496).
	nthread number of threads for OpenMP run (default: 1).
	print.progress a logical to show current iteration (default: `TRUE`).

Value

a `"histogram"` object of barycenter.

References

Benamou J, Carlier G, Cuturi M, Nenna L, Peyré G (2015). "Iterative Bregman Projections for Regularized Transportation Problems." *SIAM Journal on Scientific Computing*, **37**(2), A1111-A1138. ISSN 1064-8275, 1095-7197, doi:10.1137/141000439.

See Also

`fbary15B`

Examples

```
#-----------------------------------------------------------------------
#                     Binned from Two Gaussians
#
# EXAMPLE : Very Small Example for CRAN; just showing how to use it!
#-----------------------------------------------------------------------
# GENERATE FROM TWO GAUSSIANS WITH DIFFERENT MEANS
set.seed(100)
x  = stats::rnorm(1000, mean=-4, sd=0.5)
y  = stats::rnorm(1000, mean=+4, sd=0.5)
bk = seq(from=-10, to=10, length.out=20)

# HISTOGRAMS WITH COMMON BREAKS
histxy = list()
histxy[[1]] = hist(x, breaks=bk, plot=FALSE)
histxy[[2]] = hist(y, breaks=bk, plot=FALSE)

# COMPUTE
hh = histbary15B(histxy, maxiter=5)
```

```
# VISUALIZE
opar <- par(no.readonly=TRUE)
par(mfrow=c(1,2))
barplot(histxy[[1]]$density, col=rgb(0,0,1,1/4),
        ylim=c(0, 0.75), main="Two Histograms")
barplot(histxy[[2]]$density, col=rgb(1,0,0,1/4),
        ylim=c(0, 0.75), add=TRUE)
barplot(hh$density, main="Barycenter",
        ylim=c(0, 0.75))
par(opar)
```

`histdist` *Distance between Histograms*

Description

Compute the p-Wasserstein distance between two 1D histograms that share the same binning, i.e., same breaks. The histograms are treated as discrete probability measures supported at bin midpoints with masses given by normalized counts. Uses the exact 1D monotone OT algorithm, not LP nor entropic regularization.

Usage

```
histdist(hist1, hist2, p = 2)
```

Arguments

`hist1`	a histogram object (class `"histogram"`).
`hist2`	a histogram object (class `"histogram"`) with the same breaks as `hist1`.
`p`	an exponent for the order of the distance (default: 2).

Value

a named list containing

distance $\mathcal{W}_p$ distance value.

Examples

```
#-----------------------------------------------------------------------
#                      Binned from Gaussian and Uniform
#
# Create two types of histograms with the same binning. One is from
# the standard normal and the other from uniform distribution in [-5,5].
#-----------------------------------------------------------------------
# GENERATE 20 HISTOGRAMS
set.seed(100)
```

```
hist20 = list()
bk = seq(from=-10, to=10, length.out=20) # common breaks
for (i in 1:10){
  hist20[[i]] = hist(stats::rnorm(100), breaks=bk, plot=FALSE)
  hist20[[i+10]] = hist(stats::runif(100, min=-5, max=5), breaks=bk, plot=FALSE)
}

# COMPUTE THE PAIRWISE DISTANCE
pdmat = array(0,c(20,20))
for (i in 1:19){
  for (j in (i+1):20){
    pdmat[i,j] = histdist(hist20[[i]], hist20[[j]], p=2)$distance
    pdmat[j,i] = pdmat[i,j]
  }
}

# VISUALIZE
opar <- par(no.readonly=TRUE)
par(pty="s")
image(pdmat, axes=FALSE, main="Pairwise 2-Wasserstein Distance between Histograms")
par(opar)
```

`histinterp` *Interpolation between Histograms*

Description

Given two histograms represented as `"histogram"` S3 objects with identical breaks, compute interpolated histograms along the 2-Wasserstein geodesic connecting them. In 1D, this is achieved by linear interpolation of quantile functions (displacement interpolation).

Usage

```
histinterp(hist1, hist2, t = 0.5, L = 2000L)
```

Arguments

`hist1`	a histogram (`"histogram"` object).
`hist2`	another histogram with the same `breaks` as `hist1`.
`t`	a scalar or numeric vector in $[0, 1]$ specifying interpolation times. `t = 0` returns `hist1`, `t = 1` returns `hist2`.
`L`	number of quantile levels used to approximate the geodesic (default: 2000). Larger `L` gives a more accurate approximation at increased computational cost.

Value

If `length(t) == 1`, a single `"histogram"` object representing the interpolated distribution at time `t`. If `length(t) > 1`, a length-`length(t)` list of `"histogram"` objects.

Examples

```
#---------------------------------------------------------------------
#                      Interpolating Two Gaussians
#
# The source histogram is created from N(-5,1/4).
# The target histogram is created from N(+5,4)
#---------------------------------------------------------------------
# SETTING
set.seed(123)
x_source = rnorm(1000, mean=-5, sd=1/2)
x_target = rnorm(1000, mean=+5, sd=2)

# BUILD HISTOGRAMS WITH COMMON BREAKS
bk = seq(from=-8, to=12, by=2)
h1 = hist(x_source, breaks=bk, plot=FALSE)
h2 = hist(x_target, breaks=bk, plot=FALSE)

# INTERPOLATE WITH 5 GRID POINTS
h_path <- histinterp(h1, h2, t = seq(0, 1, length.out = 8))

# VISUALIZE
y_slim <- c(0, max(h1$density, h2$density)) # shared y-limit
xt     <- round(h1$mids, 1) # x-ticks

opar <- par(no.readonly = TRUE)
par(mfrow = c(2,4), pty = "s")
for (i in 1:8){
  if (i < 2){
    barplot(h_path[[i]]$density,  names.arg=xt, ylim=y_slim,
            main="Source", col=rgb(0,0,1,1/4))
  } else if (i > 7){
    barplot(h_path[[i]]$density,  names.arg=xt, ylim=y_slim,
            main="Target", col=rgb(1,0,0,1/4))
  } else {
    barplot(h_path[[i]]$density,  names.arg=xt, ylim=y_slim,
            col="gray90", main=sprintf("t = %.3f", (i-1)/7))
  }
}
par(opar)
```

`histmed` *Wasserstein Median of Histograms*

Description

Given multiple histograms represented as `"histogram"` S3 objects with common breaks, compute their Fréchet (geometric) median under the 2-Wasserstein distance. In 1D, this is implemented by mapping histograms to their quantile functions and running a Weiszfeld-type algorithm for the geometric median in the Hilbert space of quantile functions.

Usage

```
histmed(hists, weights = NULL, L = 2000L, ...)
```

Arguments

hists	a length-N list of histograms ("histogram" objects) with identical breaks.
weights	a weight for each histogram; if NULL (default), uniform weights are used. Otherwise, it should be a length-N vector of nonnegative weights.
L	number of quantile levels used to approximate the median (default: 2000). Larger L gives a more accurate approximation at increased computational cost.
...	extra parameters including **abstol** stopping criterion for iterations (default: 1e-8). **maxiter** maximum number of iterations (default: 496). **print.progress** logical; whether to show current iteration (default: FALSE).

Value

a "histogram" object representing the Wasserstein median.

Examples

```
#---------------------------------------------------------------------
#                      Binned from Two Gaussians
#
# Generate 12 histograms from N(-4,1/4) and 8 from N(4,1/4)
#---------------------------------------------------------------------
# COMMON SETTING
set.seed(100)
bk = seq(from=-10, to=10, length.out=20)
n_signal = 12

# GENERATE HISTOGRAMS WITH COMMON BREAKS
hist_all = list()
for (i in 1:n_signal){
  hist_all[[i]] = hist(stats::rnorm(200, mean=-4, sd=0.5), breaks=bk)
}
for (j in (n_signal+1):20){
  hist_all[[j]] = hist(stats::rnorm(200, mean=+4, sd=0.5), breaks=bk)
}

# COMPUTE THE BARYCENTER AND THE MEDIAN
h_bary = histbary(hist_all)
h_med  = histmed(hist_all)

# VISUALIZE
xt   <- round(h_med$mids, 1)
opar <- par(no.readonly=TRUE)
par(mfrow=c(1,3), pty="s")
barplot(hist_all[[1]]$density, col=rgb(0,0,1,1/4),
```

```
        ylim=c(0, 0.75), main="Two Types", names.arg=xt)
barplot(hist_all[[20]]$density, col=rgb(1,0,0,1/4),
        ylim=c(0, 0.75), add=TRUE)
barplot(h_med$density,  names.arg=xt, main="Median", ylim=c(0, 0.75))
barplot(h_bary$density, names.arg=xt, main="Barycenter", ylim=c(0, 0.75))
par(opar)
```

`imagebary` *Barycenter of Images*

Description

Using exact balanced optimal transport as a subroutine, `imagebary` computes an unregularized 2-Wasserstein barycenter image $X^\star$ from multiple input images $X_1, \ldots, X_N$. Unlike the other image barycenter routines, this function does not use entropic regularization. Instead, it solves the barycenter problem with a robust first-order method based on mirror descent on the probability simplex.

Usage

```
imagebary(images, p = 2, weights = NULL, C = NULL, ...)
```

Arguments

`images`	a length-N list of same-size image matrices of size $(m \times n)$.
`p`	an exponent for the order of the distance (default: 2). Currently, only `p=2` is supported (squared ground distance cost).
`weights`	a weight of each image; if `NULL` (default), uniform weight is set. Otherwise, it should be a length-N vector of nonnegative weights.
`C`	an optional $(mn \times mn)$ ground cost matrix. If `NULL` (default), the squared Euclidean grid cost is used. Providing `C` allows using alternative ground costs (e.g., geodesic distances on a manifold discretization).
`...`	extra parameters including

abstol stopping criterion based on ℓ_2 change of iterates (default: `1e-7`).

init.image an initial barycenter image (default: arithmetic mean of normalized inputs).

maxiter maximum number of mirror descent iterations (default: `200`).

step0 initial stepsize for mirror descent (default: `0.5`).

stepschedule stepsize schedule; `"sqrt"` uses $\eta_t = \text{step0}/\sqrt{t}$, and `"const"` uses $\eta_t = \text{step0}$ (default: `"sqrt"`).

eps positivity floor for the barycenter and inputs; values are truncated below `eps` and renormalized (default: `1e-15`). Larger values can improve robustness.

smooth optional mixing weight toward uniform distribution after each update, used to prevent near-zero support that may cause OT infeasibility (default: `1e-12`). Set to `0` to disable.

clip ℓ_∞ clipping threshold for the subgradient to stabilize exponentials in the KL update (default: `50`). Set to `Inf` to disable.

max_backtrack maximum number of backtracking halvings of the stepsize when an OT probe fails at the proposed update (default: `8`).

print.progress a logical to show iteration diagnostics (default: `FALSE`).

Details

The algorithm treats each image as a discrete probability distribution on a common $(m \times n)$ grid. At each iteration, it computes exact OT dual potentials u_i between the current barycenter iterate and each input image via `util_dual_emd_C`. These dual potentials form a valid subgradient of the barycenter objective, and a KL-mirror descent step produces a strictly positive update of the barycenter weights. For numerical stability, the implementation includes (i) centering of dual potentials (shift invariance), (ii) gradient clipping, (iii) log-domain normalization, and (iv) optional smoothing/backtracking safeguards to avoid infeasible OT calls.

Value

an $(m \times n)$ matrix of the barycentric image.

Examples

```
## Not run:
#-----------------------------------------------------------------------
#                       MNIST Data with Digit 3
#
# small example to compare the un- and regularized problem solutions
# choose only 10 images and run for 20 iterations with default penalties
#-----------------------------------------------------------------------
# LOAD DATA
set.seed(11)
data(digit3)
dat_small = digit3[sample(1:2000, 10)]

# RUN
run_exact = imagebary(dat_small, maxiter=20)
run_reg14 = imagebary14C(dat_small, maxiter=20)
run_reg15 = imagebary15B(dat_small, maxiter=20)

# VISUALIZE
opar <- par(no.readonly=TRUE)
par(mfrow=c(1,3), pty="s")
image(run_exact, axes=FALSE, main="Unregularized")
image(run_reg14, axes=FALSE, main="Cuturi & Doucet (2014)")
image(run_reg15, axes=FALSE, main="Benamou et al. (2015)")
par(opar)

## End(Not run)
```

`imagebary14C` *Barycenter of Images according to Cuturi & Doucet (2014)*

Description

Using entropic regularization for Wasserstein barycenter computation, `imagebary14C` finds a *barycentric* image X^* given multiple images $X_1, X_2, \ldots, X_N$. Please note the followings; (1) we only take a matrix as an image so please make it grayscale if not, (2) all images should be of same size - no resizing is performed.

Usage

```
imagebary14C(images, p = 2, weights = NULL, lambda = NULL, ...)
```

Arguments

`images`	a length-N list of same-size image matrices of size $(m \times n)$.
`p`	an exponent for the order of the distance (default: 2).
`weights`	a weight of each image; if `NULL` (default), uniform weight is set. Otherwise, it should be a length-N vector of nonnegative weights.
`lambda`	a regularization parameter; if `NULL` (default), a paper's suggestion would be taken, or it should be a nonnegative real number.
`...`	extra parameters including
	abstol stopping criterion for iterations (default: 1e-8).
	init.image an initial weight image (default: uniform weight).
	maxiter maximum number of iterations (default: 496).
	nthread number of threads for OpenMP run (default: 1).
	print.progress a logical to show current iteration (default: `TRUE`).

Value

an $(m \times n)$ matrix of the barycentric image.

References

Cuturi M, Doucet A (2014-06-22/2014-06-24). "Fast Computation of Wasserstein Barycenters." In Xing EP, Jebara T (eds.), *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, 685–693.

See Also

`fbary14C`

Examples

```
## Not run:
#------------------------------------------------------------------------
#                       MNIST Data with Digit 3
#
# EXAMPLE 1 : Very Small  Example for CRAN; just showing how to use it!
# EXAMPLE 2 : Medium-size Example for Evolution of Output
#------------------------------------------------------------------------
# EXAMPLE 1
data(digit3)
datsmall = digit3[1:2]
outsmall = imagebary14C(datsmall, maxiter=3)

# EXAMPLE 2 : Barycenter of 100 Images
# RANDOMLY SELECT THE IMAGES
data(digit3)
dat2 = digit3[sample(1:2000, 100)]  # select 100 images

# RUN SEQUENTIALLY
run10 = imagebary14C(dat2, maxiter=10)                  # first 10 iterations
run20 = imagebary14C(dat2, maxiter=10, init.image=run10) # run 40 more
run50 = imagebary14C(dat2, maxiter=30, init.image=run20) # run 50 more

# VISUALIZE
opar <- par(no.readonly=TRUE)
par(mfrow=c(2,3), pty="s")
image(dat2[[sample(100,1)]], axes=FALSE, main="a random image")
image(dat2[[sample(100,1)]], axes=FALSE, main="a random image")
image(dat2[[sample(100,1)]], axes=FALSE, main="a random image")
image(run10, axes=FALSE, main="barycenter after 10 iter")
image(run20, axes=FALSE, main="barycenter after 20 iter")
image(run50, axes=FALSE, main="barycenter after 50 iter")
par(opar)

## End(Not run)
```

`imagebary15B` *Barycenter of Images according to Benamou et al. (2015)*

Description

Using entropic regularization for Wasserstein barycenter computation, `imagebary15B` finds a *barycentric* image X^* given multiple images $X_1, X_2, \ldots, X_N$. Please note the followings; (1) we only take a matrix as an image so please make it grayscale if not, (2) all images should be of same size - no resizing is performed.

Usage

```
imagebary15B(images, p = 2, weights = NULL, lambda = NULL, ...)
```

Arguments

images	a length-N list of same-size image matrices of size $(m \times n)$.
p	an exponent for the order of the distance (default: 2).
weights	a weight of each image; if NULL (default), uniform weight is set. Otherwise, it should be a length-N vector of nonnegative weights.
lambda	a regularization parameter; if NULL (default), a paper's suggestion would be taken, or it should be a nonnegative real number.
...	extra parameters including
	abstol stopping criterion for iterations (default: 1e-8).
	init.image an initial weight image (default: uniform weight).
	maxiter maximum number of iterations (default: 496).
	nthread number of threads for OpenMP run (default: 1).
	print.progress a logical to show current iteration (default: TRUE).

Value

an $(m \times n)$ matrix of the barycentric image.

References

Benamou J, Carlier G, Cuturi M, Nenna L, Peyré G (2015). "Iterative Bregman Projections for Regularized Transportation Problems." *SIAM Journal on Scientific Computing*, **37**(2), A1111-A1138. ISSN 1064-8275, 1095-7197, doi:10.1137/141000439.

See Also

fbary15B

Examples

```
#---------------------------------------------------------------------
#                       MNIST Data with Digit 3
#
# EXAMPLE 1 : Very Small  Example for CRAN; just showing how to use it!
# EXAMPLE 2 : Medium-size Example for Evolution of Output
#---------------------------------------------------------------------
# EXAMPLE 1
data(digit3)
datsmall = digit3[1:2]
outsmall = imagebary15B(datsmall, maxiter=3)

## Not run:
# EXAMPLE 2 : Barycenter of 100 Images
# RANDOMLY SELECT THE IMAGES
data(digit3)
dat2 = digit3[sample(1:2000, 100)]  # select 100 images

# RUN SEQUENTIALLY
```

```
run05 = imagebary15B(dat2, maxiter=5)                    # first 5 iterations
run10 = imagebary15B(dat2, maxiter=5,  init.image=run05) # run 5 more
run50 = imagebary15B(dat2, maxiter=40, init.image=run10) # run 40 more

# VISUALIZE
opar <- par(no.readonly=TRUE)
par(mfrow=c(2,3), pty="s")
image(dat2[[sample(100,1)]], axes=FALSE, main="a random image")
image(dat2[[sample(100,1)]], axes=FALSE, main="a random image")
image(dat2[[sample(100,1)]], axes=FALSE, main="a random image")
image(run05, axes=FALSE, main="barycenter after 05 iter")
image(run10, axes=FALSE, main="barycenter after 10 iter")
image(run50, axes=FALSE, main="barycenter after 50 iter")
par(opar)

## End(Not run)
```

`imagedist` *Wasserstein Distance between Two Images*

Description

Given two grayscale images represented as numeric matrices, compute their Wasserstein distance using an exact balanced optimal transport solver. Each image is interpreted as a discrete probability distribution on a common $(m \times n)$ grid. The ground cost is defined using the Euclidean distance between grid locations.

Usage

```
imagedist(x, y, p = 2)
```

Arguments

`x`	a grayscale image matrix of size $(m \times n)$ with nonnegative entries.
`y`	a grayscale image matrix of size $(m \times n)$ with nonnegative entries.
`p`	an exponent for the order of the distance (default: 2).

Value

a list containing

distance the Wasserstein distance $W_p(x, y)$.

plan the optimal transport plan matrix of size $(mn \times mn)$.

Examples

```
#---------------------------------------------------------------------
#                       Small MNIST-like Example
#---------------------------------------------------------------------
# DATA
data(digit3)
x <- digit3[[1]]
y <- digit3[[2]]

# COMPUTE
W1 <- imagedist(x, y, p=1)
W2 <- imagedist(x, y, p=2)

# SHOW RESULTS
print(paste0("Wasserstein-1 distance: ", round(W1$distance,4)))
print(paste0("Wasserstein-2 distance: ", round(W2$distance,4)))
```

`imageinterp` *Interpolation between Images*

Description

Given two grayscale images represented as numeric matrices of identical size, compute interpolated images along a 2-Wasserstein geodesic connecting them. The function interprets each image as a discrete probability distribution on a common $(m \times n)$ grid, computes an exact optimal transport plan, and constructs intermediate measures by pushing the plan through the linear interpolation map $z = (1-t)x + ty$ (displacement interpolation / McCann's interpolation).

Usage

```
imageinterp(image1, image2, t = 0.5, ...)
```

Arguments

`image1`	a grayscale image matrix of size $(m \times n)$ with nonnegative entries.
`image2`	another grayscale image matrix of size $(m \times n)$ with nonnegative entries.
`t`	a scalar or numeric vector in $[0, 1]$ specifying interpolation times. `t=0` returns `image1`, `t=1` returns `image2`.
`...`	extra parameters including
	eps positivity floor applied after normalization (default: `1e-15`). Larger values can improve robustness.
	abstol tolerance used for internal mass checks (default: `1e-12`).
	print.progress logical; if `TRUE`, print basic diagnostics (default: `FALSE`).

Details

Because the interpolated support locations generally do not coincide with the original grid points, the resulting distribution is projected back onto the grid by depositing transported mass to the nearest grid location. This is a simple and robust "re-binning" step, analogous in spirit to how `histinterp` re-bins interpolated quantile samples.

Value

If `length(t)==1`, a single $(m \times n)$ matrix representing the interpolated image. If `length(t)>1`, a length-`length(t)` list of $(m \times n)$ matrices.

Examples

```
#---------------------------------------------------------------------
#                  Digit Interpolation between 1 and 8
#---------------------------------------------------------------------
# LOAD DATA
set.seed(11)
data(digits)
x1 <- digits$image[[sample(which(digits$label==1),1)]]
x2 <- digits$image[[sample(which(digits$label==8),1)]]

# COMPUTE
tvec <- seq(0, 1, length.out=10)
path <- imageinterp(x1, x2, t = tvec)

# VISUALIZE
opar <- par(no.readonly=TRUE)
par(mfrow=c(2,5), pty="s")
for (k in 1:10){
  image(path[[k]], axes=FALSE, main=sprintf("t=%.2f", tvec[k]))
}
par(opar)
```

---

imagemed *Wasserstein Median of Images*

Description

Using exact balanced optimal transport as a subroutine, `imagemed` computes an unregularized 2-Wasserstein geometric median image $X^\dagger$ from multiple input images $X_1, \ldots, X_N$. The Wasserstein median is defined as a minimizer of the (weighted) sum of Wasserstein distances,

$$\arg\min_X \sum_{i=1}^{N} w_i \, W_2(X, X_i).$$

Usage

```
imagemed(images, weights = NULL, C = NULL, ...)
```

Arguments

`images`	a length-N list of same-size grayscale image matrices of size $(m \times n)$.
`weights`	a weight of each image; if `NULL` (default), uniform weight is set. Otherwise, it should be a length-N vector of nonnegative weights.
`C`	an optional $(mn \times mn)$ ground cost matrix (squared distances). If `NULL` (default), the squared Euclidean grid cost is used.
`...`	extra parameters including

maxiter maximum number of IRLS outer iterations (default: `30`).

abstol stopping tolerance based on ℓ_2 change of iterates (default: `1e-6`).

delta small positive number to avoid division by zero in IRLS weights (default: `1e-8`).

init.image initial median iterate (default: unweighted barycenter via `imagebary` with a small number of iterations).

init.bary.iter iterations for the default initialization barycenter (default: `10`).

bary.maxiter maximum iterations for each barycenter subproblem (default: `200`).

bary.abstol tolerance for each barycenter subproblem (default: `1e-7`).

bary.step0 initial step size for barycenter subproblem (default: `0.5`).

bary.stepschedule `"sqrt"` or `"const"` for barycenter subproblem (default: `"sqrt"`).

bary.eps positivity floor used inside barycenter (default: `1e-15`).

bary.smooth smoothing used inside barycenter (default: `1e-12`).

bary.clip gradient clipping used inside barycenter (default: `50`).

bary.max_backtrack backtracking cap used inside barycenter (default: `8`).

print.progress logical; if `TRUE`, print iteration diagnostics (default: `FALSE`).

Details

Unlike Wasserstein barycenters (which minimize squared distances), the median is a robust notion of centrality. This function solves the problem with an iterative reweighted least squares (IRLS) scheme (a Wasserstein analogue of Weiszfeld's algorithm). Each outer iteration updates weights based on current distances and then solves a weighted Wasserstein barycenter problem:

$$\alpha_i^{(k)} \propto \frac{w_i}{\max(W_2(X^{(k)}, X_i), \delta)}, \qquad X^{(k+1)} = \arg\min_X \sum_{i=1}^{N} \alpha_i^{(k)} W_2^2(X, X_i).$$

The barycenter subproblem is solved by `imagebary` (mirror descent with exact OT dual subgradients). Distances W_2 are computed by exact EMD plans under the same squared ground cost.

Value

an $(m \times n)$ matrix of the median.

Examples

```
## Not run:
#-----------------------------------------------------------------------
#                              MNIST Example
#
# Use 6 images from digit '8' and 4 images from digit '1'.
# The median should look closer to the shape of '8'.
#-----------------------------------------------------------------------
# DATA PREP
set.seed(11)
data(digits)
dat_8 = digits$image[sample(which(digits$label==8), 6)]
dat_1 = digits$image[sample(which(digits$label==1), 4)]
dat_all = c(dat_8, dat_1)

# COMPUTE BARYCENTER AND MEDIAN
img_bary = imagebary(dat_all, maxiter=50)
img_med  = imagemed(dat_all, maxiter=50)

# VISUALIZE
opar <- par(no.readonly=TRUE)
par(mfrow=c(1,2), pty="s")
image(img_bary, axes=FALSE, main="Barycenter")
image(img_med,  axes=FALSE, main="Median")
par(opar)

## End(Not run)
```

`img2measure` *Extract a discrete measure from a gray-scale image matrix*

Description

This function takes a gray-scale image represented as a matrix X and converts it into a discrete measure suitable for optimal transport computations in a Lagrangian framework. Pixel intensities are normalized to sum to one, and the nonzero pixels are represented as weighted points (support and weights).

Usage

```
img2measure(X, threshold = TRUE)
```

Arguments

`X`	An $(N, 2)$ nonnegative matrix representing a gray-scale image, where each entry corresponds to a pixel intensity.
`threshold`	A logical flag indicating whether to threshold very small weights smaller than machine epsilon.

Value

A named list containing

support an $(M \times 2)$ matrix of coordinates for the nonzero pixels, where each row is a point (x, y).

weight a length-M vector of weights corresponding to the nonzero pixels, summing to 1.

Examples

```
#-----------------------------------------------------------------
#                          Description
#
# Take a digit image and compare visualization.
#-----------------------------------------------------------------
# load the data and select the first image
data(digit3)
img_matrix = digit3[[1]]

# extract a discrete measure
img_measure = img2measure(img_matrix, threshold=TRUE)
w  <- img_measure$weight
w_norm <- w / max(w)          # now runs from 0 to 1
col_scale <- gray(1 - w_norm) # 1 = white, 0 = black

# visualize
opar <- par(no.readonly=TRUE)
par(mfrow=c(1,2), pty="s")
image(img_matrix, xaxt="n", yaxt="n", main="Image Matrix")
plot(img_measure$support,
     col = col_scale, xlab="", ylab="",
     pch = 19, cex = 0.5, xaxt = "n", yaxt = "n",
     main = "Extracted Discrete Measure")
par(opar)
```

---

ipot *Wasserstein Distance via Inexact Proximal Point Method*

Description

The Inexact Proximal Point Method (IPOT) offers a computationally efficient approach to approximating the Wasserstein distance between two empirical measures by iteratively solving a series of regularized optimal transport problems. This method replaces the entropic regularization used in Sinkhorn's algorithm with a proximal formulation that avoids the explicit use of entropy, thereby mitigating numerical instabilities.

Let $C := \|X_m - Y_n\|^p$ be the cost matrix, where X_m and Y_n are the support points of two discrete distributions μ and ν, respectively. The IPOT algorithm solves a sequence of optimization problems:

$$\Gamma^{(t+1)} = \arg\min_{\Gamma \in \Pi(\mu,\nu)} \langle \Gamma, C \rangle + \lambda D(\Gamma \| \Gamma^{(t)}),$$

where $\lambda > 0$ is the proximal regularization parameter and $D(\cdot\|\cdot)$ is the Kullback–Leibler divergence. Each subproblem is solved approximately using a fixed number of inner iterations, making the method inexact.

Unlike entropic methods, IPOT does not require $\lambda \to 0$ for convergence to the unregularized Wasserstein solution. It is therefore more robust to numerical precision issues, especially for small regularization parameters, and provides a closer approximation to the true optimal transport cost with fewer artifacts.

Usage

```
ipot(X, Y, p = 2, wx = NULL, wy = NULL, lambda = 1, ...)

ipotD(D, p = 2, wx = NULL, wy = NULL, lambda = 1, ...)
```

Arguments

`X`	an $(M \times P)$ matrix of row observations.
`Y`	an $(N \times P)$ matrix of row observations.
`p`	an exponent for the order of the distance (default: 2).
`wx`	a length-M marginal density that sums to 1. If `NULL` (default), uniform weight is set.
`wy`	a length-N marginal density that sums to 1. If `NULL` (default), uniform weight is set.
`lambda`	a regularization parameter (default: 0.1).
`...`	extra parameters including **maxiter** maximum number of iterations (default: 496). **abstol** stopping criterion for iterations (default: 1e-10). **L** small number of inner loop iterations (default: 1).
`D`	an $(M \times N)$ distance matrix $d(x_m, y_n)$ between two sets of observations.

Value

a named list containing

distance $\mathcal{W}_p$ distance value

plan an $(M \times N)$ nonnegative matrix for the optimal transport plan.

References

Xie Y, Wang X, Wang R, Zha H (2020-07-22/2020-07-25). "A Fast Proximal Point Method for Computing Exact Wasserstein Distance." In Adams RP, Gogate V (eds.), *Proceedings of the 35th Uncertainty in Artificial Intelligence Conference*, volume 115 of *Proceedings of Machine Learning Research*, 433–453.

Examples

```
#-----------------------------------------------------------------
#  Wasserstein Distance between Samples from Two Bivariate Normal
#
# * class 1 : samples from Gaussian with mean=(-1, -1)
# * class 2 : samples from Gaussian with mean=(+1, +1)
#-----------------------------------------------------------------
## SMALL EXAMPLE
set.seed(100)
m = 20
n = 30
X = matrix(rnorm(m*2, mean=-1),ncol=2) # m obs. for X
Y = matrix(rnorm(n*2, mean=+1),ncol=2) # n obs. for Y

## COMPARE WITH WASSERSTEIN
outw = wasserstein(X, Y)
ipt1 = ipot(X, Y, lambda=1)
ipt2 = ipot(X, Y, lambda=10)

## VISUALIZE : SHOW THE PLAN AND DISTANCE
pmw = paste0("Exact plan\n dist=",round(outw$distance,2))
pm1 = paste0("IPOT (lambda=1)\n dist=",round(ipt1$distance,2))
pm2 = paste0("IPOT (lambda=10)\n dist=",round(ipt2$distance,2))

opar <- par(no.readonly=TRUE)
par(mfrow=c(1,3), pty="s")
image(outw$plan, axes=FALSE, main=pmw)
image(ipt1$plan, axes=FALSE, main=pm1)
image(ipt2$plan, axes=FALSE, main=pm2)
par(opar)
```

---

pwbary *Procrustes-Wasserstein Barycenter*

Description

For a collection of empirical measures $\{\mu_k\}_{k=1}^K$, this function computes the Procrustes-Wasserstein (PW) barycenter (Adamo et al. 2025), which accounts for both measure transport and alignment through action of the orthogonal group.

Usage

```
pwbary(atoms, marginals = NULL, weights = NULL, num_support = 100, ...)
```

Arguments

`atoms`	a length-K list where each element is an $(N_k \times P)$ matrix of atoms.
`marginals`	marginal distributions for empirical measures; if `NULL` (default), uniform weights are set for all measures. Otherwise, it should be a length-K list where each element is a length-N_i vector of nonnegative weights that sum to 1.
`weights`	weights for each individual measure; if `NULL` (default), each measure is considered equally. Otherwise, it should be a length-K vector.
`num_support`	the number of support points M for the PW barycenter (default: 100).
`...`	extra parameters including
	abstol stopping criterion for iterations (default: 1e-6).
	maxiter maximum number of iterations (default: 10).

Value

a list with three elements:

support an $(M \times P)$ matrix of the PW barycenter's support points.

weight a length-M vector of median's weights with all entries being $1/M$.

References

Adamo D, Corneli M, Vuillien M, Vila E (2025). "An in Depth Look at the Procrustes-Wasserstein Distance: Properties and Barycenters." In *Forty-Second International Conference on Machine Learning*.

Examples

```
## Not run:
#------------------------------------------------------------------
#         Free-Support PW Barycenter of Multiple Gaussians
#
# * class 1 : samples from N((0,0),  diag(c(4,1/4)))
# * class 2 : samples from N((10,0), diag(c(1/4,4)))
# * class 3 : samples from N((10,0), Id) randomly rotated
#
#  We draw 10 empirical measures from each and compare
#  their barycenters under the regular and PW geometries.
#------------------------------------------------------------------
## GENERATE DATA
set.seed(10)

#  prepare empty lists
input_1 = vector("list", length=10L)
input_2 = vector("list", length=10L)
input_3 = vector("list", length=10L)

#  generate
random_rot = qr.Q(qr(matrix(runif(4), ncol=2)))
for (i in 1:10){
```

```
  input_1[[i]] = cbind(rnorm(50, sd=2), rnorm(50, sd=0.5))
}
for (j in 1:10){
  base_draw = cbind(rnorm(50, sd=0.5), rnorm(50, sd=2))
  base_draw[,1] = base_draw[,1] + 10

  input_2[[j]] = base_draw
  input_3[[j]] = base_draw%*%random_rot
}

## COMPUTE
#  regular Wasserstein barycenters
regular_1 = rbaryGD(input_1, num_support=50)
regular_2 = rbaryGD(input_2, num_support=50)
regular_3 = rbaryGD(input_3, num_support=50)

#  Procrustes-Wasserstein barycenters
pw_1 = pwbary(input_1, num_support=50)
pw_2 = pwbary(input_2, num_support=50)
pw_3 = pwbary(input_3, num_support=50)

## VISUALIZE
opar <- par(no.readonly=TRUE)
par(mfrow=c(3,1))

#  set the x- and y-limits for display
lim_x = c(-12, 12)
lim_y = c(-10, 5)

#  plot prototypical measures per class
plot(input_1[[1]], pch=19, cex=0.5, col="gray80",
     main="3 types of measures", xlab="", ylab="",
     xlim=lim_x, ylim=lim_y)
points(input_2[[1]], pch=19, cex=0.5, col="gray50")
points(input_3[[1]], pch=19, cex=0.5, col="gray10")

#  plot regular barycenters
plot(regular_1$support, pch=19, cex=0.5, col="blue",
     main="Regular Wasserstein barycenters",
     xlab="", ylab="", xlim=lim_x, ylim=lim_y)
points(regular_2$support, pch=19, cex=0.5, col="cyan")
points(regular_3$support, pch=19, cex=0.5, col="red")

#  plot PW barycenters
plot(pw_1$support, pch=19, cex=0.5, col="blue",
     main="Procrustes-Wasserstein barycenters",
     xlab="", ylab="", xlim=lim_x, ylim=lim_y)
points(pw_2$support, pch=19, cex=0.5, col="cyan")
points(pw_3$support, pch=19, cex=0.5, col="red")
par(opar)

## End(Not run)
```

pwdist *Procrustes-Wasserstein Distance*

Description

Given two empirical measures

$$\mu = \sum_{m=1}^{M} \mu_m \delta_{X_m} \quad \text{and} \quad \nu = \sum_{n=1}^{N} \nu_n \delta_{Y_n}$$

in $\mathbb{R}^P$, the Procrustes-Wasserstein (PW) distance is defined as follows:

$$PW_2^2(\mu, \nu) = \min_{Q \in \mathcal{O}(P)} W_2^2(\mu, Q_{\#}\nu),$$

where $\mathcal{O}(P)$ is the orthogonal group and $Q_{\#}\nu$ is the pushforward via Q.

Usage

```
pwdist(X, Y, wx = NULL, wy = NULL, ...)
```

Arguments

X	an $(M \times P)$ matrix of row observations.
Y	an $(N \times P)$ matrix of row observations.
wx	a length-M marginal density that sums to 1. If NULL (default), uniform weight is set.
wy	a length-N marginal density that sums to 1. If NULL (default), uniform weight is set.
...	extra parameters including **maxiter** maximum number of iterations (default: 496). **abstol** stopping criterion for iterations (default: 1e-10).

Value

a named list containing

distance the computed PW distance value.

plan an $(M \times N)$ nonnegative matrix for the optimal transport plan.

alignment an optimal alignment matrix of size $(P \times P)$ in $\mathcal{O}(P)$.

References

Adamo D, Corneli M, Vuillien M, Vila E (2025). "An in Depth Look at the Procrustes-Wasserstein Distance: Properties and Barycenters." In *Forty-Second International Conference on Machine Learning*.

Examples

```
## Not run:
#-----------------------------------------------------------------------
#                           Description
#
# * class 1 : samples from N((0,0),  diag(c(4,1/4)))
# * class 2 : samples from N((10,0), diag(c(1/4,4)))
# * class 3 : samples from N((10,0), diag(c(1/4,4))) randomly rotated
#
#  We draw 10 empirical measures from each and compare
#  the regular Wasserstein and PW distance.
#-----------------------------------------------------------------------
## GENERATE DATA
set.seed(10)

#  prepare empty lists
inputs = vector("list", length=30)

#  generate
random_rot = qr.Q(qr(matrix(runif(4), ncol=2)))
for (i in 1:10){
  inputs[[i]] = matrix(rnorm(50*2), ncol=2)
}
for (j in 11:20){
  base_draw = matrix(rnorm(50*2), ncol=2)
  base_draw[,1] = base_draw[,1] + 10

  inputs[[j]] = base_draw
  inputs[[j+10]] = base_draw%*%random_rot
}

## COMPUTE
#  empty arrays
dist_RW = array(0, c(30, 30))
dist_PW = array(0, c(30, 30))

#  compute pairwise distances
for (i in 1:29){
  for (j in (i+1):30){
  dist_RW[i,j] <- dist_RW[j,i] <- wasserstein(inputs[[i]], inputs[[j]])$distance
  dist_PW[i,j] <- dist_PW[j,i] <- pwdist(inputs[[i]], inputs[[j]])$distance
  }
}

## VISUALIZE
opar <- par(no.readonly=TRUE)
par(mfrow=c(1,2), pty="s")
image(dist_RW, xaxt="n", yaxt="n", main="Regular Wasserstein distance")
image(dist_PW, xaxt="n", yaxt="n", main="PW distance")
par(opar)

## End(Not run)
```

rbary23L *Free-Support Barycenter by von Lindheim (2023)*

Description

For a collection of empirical measures $\{\mu_k\}_{k=1}^K$, this function implements the free-support barycenter algorithm introduced by von Lindheim (2023). The algorithm takes the first input and its marginal as a reference and performs one-step update of the support. This version implements 'reference' algorithm with $p = 2$.

Usage

```
rbary23L(atoms, marginals = NULL, weights = NULL)
```

Arguments

`atoms`	a length-K list where each element is an $(N_k \times P)$ matrix of atoms.
`marginals`	marginal distributions for empirical measures; if `NULL` (default), uniform weights are set for all measures. Otherwise, it should be a length-K list where each element is a length-N_i vector of nonnegative weights that sum to 1.
`weights`	weights for each individual measure; if `NULL` (default), each measure is considered equally. Otherwise, it should be a length-K vector.

Value

a list with two elements:

support an $(N_1 \times P)$ matrix of barycenter support points (same number of atoms as the first empirical measure).

weight a length-N_1 vector representing barycenter weights (copied from the first marginal).

References

von Lindheim J (2023). "Simple Approximative Algorithms for Free-Support Wasserstein Barycenters." *Computational Optimization and Applications*, **85**(1), 213–246. ISSN 0926-6003, 1573-2894, doi:10.1007/s10589023004583.

Examples

```
#----------------------------------------------------------------
#     Free-Support Wasserstein Barycenter of Four Gaussians
#
# * class 1 : samples from Gaussian with mean=(-4, -4)
# * class 2 : samples from Gaussian with mean=(+4, +4)
# * class 3 : samples from Gaussian with mean=(+4, -4)
# * class 4 : samples from Gaussian with mean=(-4, +4)
```

```
#
#  The barycenter is computed using the first measure as a reference.
#  All measures have uniform weights.
#  The barycenter function also considers uniform weights.
#---------------------------------------------------------------------
## GENERATE DATA
#  Empirical Measures
set.seed(100)
unif4 = round(runif(4, 100, 200))
dat1 = matrix(rnorm(unif4[1]*2, mean=-4, sd=0.5),ncol=2)
dat2 = matrix(rnorm(unif4[2]*2, mean=+4, sd=0.5),ncol=2)
dat3 = cbind(rnorm(unif4[3], mean=+4, sd=0.5), rnorm(unif4[3], mean=-4, sd=0.5))
dat4 = cbind(rnorm(unif4[4], mean=-4, sd=0.5), rnorm(unif4[4], mean=+4, sd=0.5))

myatoms = list()
myatoms[[1]] = dat1
myatoms[[2]] = dat2
myatoms[[3]] = dat3
myatoms[[4]] = dat4

## COMPUTE
fsbary = rbary23L(myatoms)

## VISUALIZE
#  aligned with CRAN convention
opar <- par(no.readonly=TRUE)

#  plot the input measures
plot(myatoms[[1]], col="gray90", pch=19, cex=0.5, xlim=c(-6,6), ylim=c(-6,6),
     main="Input Measures", xlab="Dimension 1", ylab="Dimension 2")
points(myatoms[[2]], col="gray90", pch=19, cex=0.25)
points(myatoms[[3]], col="gray90", pch=19, cex=0.25)
points(myatoms[[4]], col="gray90", pch=19, cex=0.25)

#  plot the barycenter
points(fsbary$support, col="red", cex=0.5, pch=19)
par(opar)
```

---

rbaryGD *Free-Support Barycenter by Riemannian Gradient Descent*

Description

For a collection of empirical measures $\{\mu_k\}_{k=1}^K$, the free-support barycenter of order 2, defined as a minimizer of the following functional,

$$\mathcal{F}(\nu) = \sum_{k=1}^{K} w_k \mathcal{W}_2^2(\nu, \mu_k),$$

is computed using the Riemannian gradient descent algorithm. The algorithm is based on the formal Riemannian geometric view of the 2-Wasserstein space according to Otto (2001).

Usage

```
rbaryGD(atoms, marginals = NULL, weights = NULL, num_support = 100, ...)
```

Arguments

`atoms`	a length-K list where each element is an $(N_k \times P)$ matrix of atoms.
`marginals`	marginal distributions for empirical measures; if `NULL` (default), uniform weights are set for all measures. Otherwise, it should be a length-K list where each element is a length-N_k vector of nonnegative weights that sum to 1.
`weights`	weights for each individual measure; if `NULL` (default), each measure is considered equally. Otherwise, it should be a length-K vector.
`num_support`	the number of support points M for the barycenter (default: 100).
`...`	extra parameters including **abstol** stopping criterion for iterations (default: 1e-6). **maxiter** maximum number of iterations (default: 10).

Value

a list with three elements:

support an $(M \times P)$ matrix of barycenter support points.

weight a length-M vector of barycenter weights with all entries being $1/M$.

history a vector of cost values at each iteration.

References

Otto F (2001). "The Geometry of Dissipative Evolution Equations: The Porous Medium Equation." *Communications in Partial Differential Equations*, **26**(1-2), 101–174. ISSN 0360-5302, 1532-4133, doi:10.1081/PDE100002243.

Examples

```
#-----------------------------------------------------------------
#     Free-Support Wasserstein Barycenter of Four Gaussians
#
# * class 1 : samples from Gaussian with mean=(-4, -4)
# * class 2 : samples from Gaussian with mean=(+4, +4)
# * class 3 : samples from Gaussian with mean=(+4, -4)
# * class 4 : samples from Gaussian with mean=(-4, +4)
#
#  All measures have uniform weights.
#-----------------------------------------------------------------
## GENERATE DATA
#  Empirical Measures
set.seed(100)
```

```
unif4 = round(runif(4, 100, 200))
dat1 = matrix(rnorm(unif4[1]*2, mean=-4, sd=0.5),ncol=2)
dat2 = matrix(rnorm(unif4[2]*2, mean=+4, sd=0.5),ncol=2)
dat3 = cbind(rnorm(unif4[3], mean=+4, sd=0.5), rnorm(unif4[3], mean=-4, sd=0.5))
dat4 = cbind(rnorm(unif4[4], mean=-4, sd=0.5), rnorm(unif4[4], mean=+4, sd=0.5))

myatoms = list()
myatoms[[1]] = dat1
myatoms[[2]] = dat2
myatoms[[3]] = dat3
myatoms[[4]] = dat4

## COMPUTE
fsbary = rbaryGD(myatoms)

## VISUALIZE
#  aligned with CRAN convention
opar <- par(no.readonly=TRUE, mfrow=c(1,2))

#  plot the input measures and the barycenter
plot(myatoms[[1]], col="gray90", pch=19, cex=0.5, xlim=c(-6,6), ylim=c(-6,6),
     main="Inputs and Barycenter", xlab="Dimension 1", ylab="Dimension 2")
points(myatoms[[2]], col="gray90", pch=19, cex=0.25)
points(myatoms[[3]], col="gray90", pch=19, cex=0.25)
points(myatoms[[4]], col="gray90", pch=19, cex=0.25)
points(fsbary$support, col="red", cex=0.5, pch=19)

#  plot the cost history with only integer ticks
plot(seq_along(fsbary$history), fsbary$history, type="b", lwd=2, pch=19,
     main="Cost History", xlab="Iteration", ylab="Cost", xaxt='n')
axis(1, at=seq_along(fsbary$history))
par(opar)
```

`rmedIRLS` *Free-Support Median by IRLS*

Description

For a collection of empirical measures $\{\mu_k\}_{k=1}^K$, the free-support Wasserstein median, a minimizer to the following functional

$$\mathcal{F}(\nu) = \sum_{k=1}^{K} w_k \mathcal{W}_2(\nu, \mu_k),$$

is computed using the generic method of iteratively-reweighted least squares (IRLS) method according to You et al. (2025).

Usage

```
rmedIRLS(atoms, marginals = NULL, weights = NULL, num_support = 100, ...)
```

Arguments

`atoms`	a length-K list where each element is an $(N_k \times P)$ matrix of atoms.
`marginals`	marginal distributions for empirical measures; if `NULL` (default), uniform weights are set for all measures. Otherwise, it should be a length-K list where each element is a length-N_i vector of nonnegative weights that sum to 1.
`weights`	weights for each individual measure; if `NULL` (default), each measure is considered equally. Otherwise, it should be a length-K vector.
`num_support`	the number of support points M for the barycenter (default: 100).
`...`	extra parameters including
	abstol stopping criterion for iterations (default: 1e-6).
	maxiter maximum number of iterations (default: 10).

Value

a list with three elements:

support an $(M \times P)$ matrix of the Wasserstein median's support points.

weight a length-M vector of median's weights with all entries being $1/M$.

history a vector of cost values at each iteration.

References

You K, Shung D, Giuffrè M (2025). "On the Wasserstein Median of Probability Measures." *Journal of Computational and Graphical Statistics*, **34**(1), 253-266. ISSN 1061-8600, 1537-2715.

Examples

```
## Not run:
#------------------------------------------------------------------
#     Free-Support Wasserstein Median of Multiple Gaussians
#
# * class 1 : samples from N((0,0),  Id)
# * class 2 : samples from N((20,0), Id)
#
#  We draw 8 empirical measures of size 50 from class 1, and
#  2 from class 2. All measures have uniform weights.
#------------------------------------------------------------------
## GENERATE DATA
#  8 empirical measures from class 1
input_measures = vector("list", length=10L)
for (i in 1:8){
  input_measures[[i]] = matrix(rnorm(50*2), ncol=2)
}
for (j in 9:10){
  base_draw = matrix(rnorm(50*2), ncol=2)
  base_draw[,1] = base_draw[,1] + 20
  input_measures[[j]] = base_draw
}
```

```
## COMPUTE
#  compute the Wasserstein median
run_median = rmedIRLS(input_measures, num_support = 50)
#  compute the Wasserstein barycenter
run_bary   = rbaryGD(input_measures, num_support = 50)

## VISUALIZE
opar <- par(no.readonly=TRUE)

#  draw the base points of two classes
base_1 = matrix(rnorm(80*2), ncol=2)
base_2 = matrix(rnorm(20*2), ncol=2)
base_2[,1] = base_2[,1] + 20
base_mat = rbind(base_1, base_2)
plot(base_mat, col="gray80", pch=19)

#  auxiliary information
title("estimated barycenter and median")
abline(v=0); abline(h=0)

#  draw the barycenter and the median
points(run_bary$support, col="red", pch=19)
points(run_median$support, col="blue", pch=19)
par(opar)

## End(Not run)
```

---

rmedWB *Free-Support Median by Weiszfeld Update with Barycentric Projection*

Description

For a collection of empirical measures $\{\mu_k\}_{k=1}^K$, the free-support Wasserstein median, a minimizer to the following functional

$$\mathcal{F}(\nu) = \sum_{k=1}^{K} w_k \mathcal{W}_2(\nu, \mu_k),$$

is computed using the OT-adapted version of the Weiszfeld algorithm using the barycentric projection as a means to recover an optimal displacement map.

Usage

```
rmedWB(atoms, marginals = NULL, weights = NULL, num_support = 100, ...)
```

Arguments

`atoms`	a length-K list where each element is an $(N_k \times P)$ matrix of atoms.
`marginals`	marginal distributions for empirical measures; if `NULL` (default), uniform weights are set for all measures. Otherwise, it should be a length-K list where each element is a length-N_i vector of nonnegative weights that sum to 1.
`weights`	weights for each individual measure; if `NULL` (default), each measure is considered equally. Otherwise, it should be a length-K vector.
`num_support`	the number of support points M for the barycenter (default: 100).
`...`	extra parameters including **abstol** stopping criterion for iterations (default: 1e-6). **maxiter** maximum number of iterations (default: 10).

Value

a list with three elements:

support an $(M \times P)$ matrix of the Wasserstein median's support points.

weight a length-M vector of median's weights with all entries being $1/M$.

history a vector of cost values at each iteration.

Examples

```
## Not run:
#-----------------------------------------------------------------------
#     Free-Support Wasserstein Median of Multiple Gaussians
#
# * class 1 : samples from N((0,0),  Id)
# * class 2 : samples from N((20,0), Id)
#
#  We draw 8 empirical measures of size 50 from class 1, and
#  2 from class 2. All measures have uniform weights.
#-----------------------------------------------------------------------
## GENERATE DATA
#  8 empirical measures from class 1
input_measures = vector("list", length=10L)
for (i in 1:8){
  input_measures[[i]] = matrix(rnorm(50*2), ncol=2)
}
for (j in 9:10){
  base_draw = matrix(rnorm(50*2), ncol=2)
  base_draw[,1] = base_draw[,1] + 20
  input_measures[[j]] = base_draw
}

## COMPUTE
#  compute the Wasserstein median
run_median = rmedWB(input_measures, num_support = 50)
#  compute the Wasserstein barycenter
run_bary   = rbaryGD(input_measures, num_support = 50)
```

```
## VISUALIZE
opar <- par(no.readonly=TRUE)

#  draw the base points of two classes
base_1 = matrix(rnorm(80*2), ncol=2)
base_2 = matrix(rnorm(20*2), ncol=2)
base_2[,1] = base_2[,1] + 20
base_mat = rbind(base_1, base_2)
plot(base_mat, col="gray80", pch=19)

#  auxiliary information
title("estimated barycenter and median")
abline(v=0); abline(h=0)

#  draw the barycenter and the median
points(run_bary$support, col="red", pch=19)
points(run_median$support, col="blue", pch=19)
par(opar)

## End(Not run)
```

`sinkhorn` *Wasserstein Distance via Entropic Regularization and Sinkhorn Algorithm*

Description

To alleviate the computational burden of solving the exact optimal transport problem via linear programming, Cuturi (2013) introduced an entropic regularization scheme that yields a smooth approximation to the Wasserstein distance. Let $C := \|X_m - Y_n\|^p$ be the cost matrix, where X_m and Y_n are the observations from two distributions μ and nu. Then, the regularized problem adds a penalty term to the objective function:

$$W^p_{p,\lambda}(\mu,\nu) = \min_{\Gamma\in\Pi(\mu,\nu)} \langle \Gamma, C\rangle + \lambda \sum_{m,n} \Gamma_{m,n}\log(\Gamma_{m,n}),$$

where $\lambda > 0$ is the regularization parameter and Γ denotes a transport plan. As $\lambda \to 0$, the regularized solution converges to the exact Wasserstein solution, but small values of λ may cause numerical instability due to underflow. In such cases, the implementation halts with an error; users are advised to increase λ to maintain numerical stability.

Usage

```
sinkhorn(X, Y, p = 2, wx = NULL, wy = NULL, lambda = 0.1, ...)

sinkhornD(D, p = 2, wx = NULL, wy = NULL, lambda = 0.1, ...)
```

Arguments

X	an $(M \times P)$ matrix of row observations.
Y	an $(N \times P)$ matrix of row observations.
p	an exponent for the order of the distance (default: 2).
wx	a length-M marginal density that sums to 1. If NULL (default), uniform weight is set.
wy	a length-N marginal density that sums to 1. If NULL (default), uniform weight is set.
lambda	a regularization parameter (default: 0.1).
...	extra parameters including
	maxiter maximum number of iterations (default: 496).
	abstol stopping criterion for iterations (default: 1e-10).
D	an $(M \times N)$ distance matrix $d(x_m, y_n)$ between two sets of observations.

Value

a named list containing

distance $\mathcal{W}_p$ distance value.

plan an $(M \times N)$ nonnegative matrix for the optimal transport plan.

References

Cuturi M (2013). "Sinkhorn Distances: Lightspeed Computation of Optimal Transport." In Burges CJ, Bottou L, Welling M, Ghahramani Z, Weinberger KQ (eds.), *Advances in Neural Information Processing Systems*, volume 26.

Examples

```
#-----------------------------------------------------------------
#  Wasserstein Distance between Samples from Two Bivariate Normal
#
# * class 1 : samples from Gaussian with mean=(-1, -1)
# * class 2 : samples from Gaussian with mean=(+1, +1)
#-----------------------------------------------------------------
## SMALL EXAMPLE
set.seed(100)
m = 20
n = 10
X = matrix(rnorm(m*2, mean=-1),ncol=2) # m obs. for X
Y = matrix(rnorm(n*2, mean=+1),ncol=2) # n obs. for Y

## COMPARE WITH WASSERSTEIN
outw = wasserstein(X, Y)
skh1 = sinkhorn(X, Y, lambda=0.05)
skh2 = sinkhorn(X, Y, lambda=0.25)

## VISUALIZE : SHOW THE PLAN AND DISTANCE
```

```
pm1 = paste0("Exact Wasserstein:\n distance=",round(outw$distance,2))
pm2 = paste0("Sinkhorn (lbd=0.05):\n distance=",round(skh1$distance,2))
pm5 = paste0("Sinkhorn (lbd=0.25):\n distance=",round(skh2$distance,2))

opar <- par(no.readonly=TRUE)
par(mfrow=c(1,3), pty="s")
image(outw$plan, axes=FALSE, main=pm1)
image(skh1$plan, axes=FALSE, main=pm2)
image(skh2$plan, axes=FALSE, main=pm5)
par(opar)
```

`swdist` *Sliced Wasserstein Distance*

Description

Sliced Wasserstein (SW) Distance is a popular alternative to the standard Wasserstein distance due to its computational efficiency on top of nice theoretical properties. For the d-dimensional probability measures μ and ν, the SW distance is defined as

$$\mathcal{SW}_p(\mu,\nu) = \left(\int_{\mathbb{S}^{d-1}} \mathcal{W}_p^p(\langle\theta,\mu\rangle,\langle\theta,\nu\rangle)d\lambda(\theta)\right)^{1/p},$$

where $\mathbb{S}^{d-1}$ is the $(d-1)$-dimensional unit hypersphere and λ is the uniform distribution on $\mathbb{S}^{d-1}$. Practically, it is computed via Monte Carlo integration.

Usage

```
swdist(X, Y, p = 2, ...)
```

Arguments

`X`	an $(M \times P)$ matrix of row observations.
`Y`	an $(N \times P)$ matrix of row observations.
`p`	an exponent for the order of the distance (default: 2).
`...`	extra parameters including
	num_proj the number of Monte Carlo samples for SW computation (default: 496).

Value

a named list containing

distance $\mathcal{SW}_p$ distance value.

projdist a length-`num_proj` vector of projected univariate distances.

References

Rabin J, Peyré G, Delon J, Bernot M (2012). "Wasserstein Barycenter and Its Application to Texture Mixing." In Bruckstein AM, ter Haar Romeny BM, Bronstein AM, Bronstein MM (eds.), *Scale Space and Variational Methods in Computer Vision*, volume 6667, 435–446. Springer Berlin Heidelberg, Berlin, Heidelberg. ISBN 978-3-642-24784-2 978-3-642-24785-9, doi:10.1007/9783-642247859_37.

Examples

```
#-----------------------------------------------------------------
#  Sliced-Wasserstein Distance between Two Bivariate Normal
#
# * class 1 : samples from Gaussian with mean=(-1, -1)
# * class 2 : samples from Gaussian with mean=(+1, +1)
#-----------------------------------------------------------------
# SMALL EXAMPLE
set.seed(100)
m = 20
n = 30
X = matrix(rnorm(m*2, mean=-1),ncol=2) # m obs. for X
Y = matrix(rnorm(n*2, mean=+1),ncol=2) # n obs. for Y

# COMPUTE THE SLICED-WASSERSTEIN DISTANCE
outsw <- swdist(X, Y, num_proj=100)

# VISUALIZE
# prepare ingredients for plotting
plot_x = 1:1000
plot_y = base::cumsum(outsw$projdist)/plot_x

# draw
opar <- par(no.readonly=TRUE)
plot(plot_x, plot_y, type="b", cex=0.1, lwd=2,
     xlab="number of MC samples", ylab="distance",
     main="Effect of MC Sample Size")
abline(h=outsw$distance, col="red", lwd=2)
legend("bottomright", legend="SW Distance",
       col="red", lwd=2)
par(opar)
```

`wassboot` *Wasserstein Distance Estimation with Boostrapping*

Description

This function computes the $\mathcal{W}_p$ distance between two empirical measures using bootstrap in order to quantify the uncertainty of the estimation.

Usage

```
wassboot(X, Y, p = 2, B = 500, wx = NULL, wy = NULL)
```

Arguments

X	an $(M \times P)$ matrix of row observations.
Y	an $(N \times P)$ matrix of row observations.
p	an exponent for the order of the distance (default: 2).
B	number of bootstrap samples (default: 500).
wx	a length-M marginal density that sums to 1. If NULL (default), uniform weight is set.
wy	a length-N marginal density that sums to 1. If NULL (default), uniform weight is set.

Value

a named list containing

distance $\mathcal{W}_p$ distance value.

boot_samples a length-B vector of bootstrap samples.

Examples

```
#-----------------------------------------------------------------
#  Boostrapping Wasserstein Distance between Two Bivariate Normals
#
# * class 1 : samples from Gaussian with mean=(-5, 0)
# * class 2 : samples from Gaussian with mean=(+5, 0)
#-----------------------------------------------------------------
## SMALL EXAMPLE
m = round(runif(1, min=50, max=100))
n = round(runif(1, min=50, max=100))
X = matrix(rnorm(m*2), ncol=2) # m obs. for X
Y = matrix(rnorm(n*2), ncol=2) # n obs. for Y

X[,1] = X[,1] - 5
Y[,1] = Y[,1] + 5

## COMPUTE THE BOOTSTRAP SAMPLES
boots = wassboot(X, Y, B=1000)

## VISUALIZE
opar <- par(no.readonly=TRUE)
hist(boots$boot_samples, xlab="Estimates", main="Bootstrap Samples")
abline(v=boots$distance, lwd=2, col="blue")
abline(v=mean(boots$boot_samples), lwd=2, col="red")
abline(v=10, col="cyan", lwd=2)
legend("topright", c("ground truth","estimate","bootstrap mean"),
        col=c("cyan","blue","red"), lwd=2)
```

```
par(opar)
```

`wasserstein` *Wasserstein Distance via Linear Programming*

Description

Given two empirical measures

$$\mu = \sum_{m=1}^{M} \mu_m \delta_{X_m} \quad \text{and} \quad \nu = \sum_{n=1}^{N} \nu_n \delta_{Y_n},$$

the p-Wasserstein distance for $p \geq 1$ is posited as the following optimization problem

$$W_p^p(\mu, \nu) = \min_{\pi \in \Pi(\mu,\nu)} \sum_{m=1}^{M} \sum_{n=1}^{N} \pi_{mn} \|X_m - Y_n\|^p,$$

where $\Pi(\mu, \nu)$ denotes the set of joint distributions (transport plans) with marginals μ and ν. This function solves the above problem with linear programming, which is a standard approach for exact computation of the empirical Wasserstein distance. Please see the section for detailed description on the usage of the function.

Usage

```
wasserstein(X, Y, p = 2, wx = NULL, wy = NULL)

wassersteinD(D, p = 2, wx = NULL, wy = NULL)
```

Arguments

`X`	an $(M \times P)$ matrix of row observations.
`Y`	an $(N \times P)$ matrix of row observations.
`p`	an exponent for the order of the distance (default: 2).
`wx`	a length-M marginal density that sums to 1. If `NULL` (default), uniform weight is set.
`wy`	a length-N marginal density that sums to 1. If `NULL` (default), uniform weight is set.
`D`	an $(M \times N)$ distance matrix $d(x_m, y_n)$ between two sets of observations.

Value

a named list containing

distance $\mathcal{W}_p$ distance value.

plan an $(M \times N)$ nonnegative matrix for the optimal transport plan.

Using `wasserstein()` function

We assume empirical measures are defined on the Euclidean space $\mathcal{X} = \mathbb{R}^d$,

$$\mu = \sum_{m=1}^{M} \mu_m \delta_{X_m} \quad \text{and} \quad \nu = \sum_{n=1}^{N} \nu_n \delta_{Y_n}$$

and the distance metric used here is standard Euclidean norm $d(x, y) = \|x-y\|$. Here, the marginals $(\mu_1, \mu_2, \ldots, \mu_M)$ and $(\nu_1, \nu_2, \ldots, \nu_N)$ correspond to `wx` and `wy`, respectively.

Using `wasserstein1D()` function

If other distance measures or underlying spaces are one's interests, we have an option for users to provide a distance matrix `D` rather than vectors, where

$$D := D_{M \times N} = d(X_m, Y_n)$$

for arbitrary distance metrics beyond the ℓ_2 norm.

References

Peyré G, Cuturi M (2019). "Computational Optimal Transport: With Applications to Data Science." *Foundations and Trends® in Machine Learning*, **11**(5-6), 355–607. ISSN 1935-8237, 1935-8245, doi:10.1561/2200000073.

Examples

```
#-----------------------------------------------------------------
#  Wasserstein Distance between Samples from Two Bivariate Normal
#
# * class 1 : samples from Gaussian with mean=(-1, -1)
# * class 2 : samples from Gaussian with mean=(+1, +1)
#-----------------------------------------------------------------
## SMALL EXAMPLE
m = 20
n = 10
X = matrix(rnorm(m*2, mean=-1),ncol=2) # m obs. for X
Y = matrix(rnorm(n*2, mean=+1),ncol=2) # n obs. for Y

## COMPUTE WITH DIFFERENT ORDERS
out1 = wasserstein(X, Y, p=1)
out2 = wasserstein(X, Y, p=2)
out5 = wasserstein(X, Y, p=5)

## VISUALIZE : SHOW THE PLAN AND DISTANCE
pm1 = paste0("Order p=1\n distance=",round(out1$distance,2))
pm2 = paste0("Order p=2\n distance=",round(out2$distance,2))
pm5 = paste0("Order p=5\n distance=",round(out5$distance,2))

opar <- par(no.readonly=TRUE)
par(mfrow=c(1,3), pty="s")
image(out1$plan, axes=FALSE, main=pm1)
```

```
image(out2$plan, axes=FALSE, main=pm2)
image(out5$plan, axes=FALSE, main=pm5)
par(opar)

## Not run:
## COMPARE WITH ANALYTIC RESULTS
#  For two Gaussians with same covariance, their
#  2-Wasserstein distance is known so let's compare !

niter = 1000          # number of iterations
vdist = rep(0,niter)
for (i in 1:niter){
  mm = sample(30:50, 1)
  nn = sample(30:50, 1)

  X = matrix(rnorm(mm*2, mean=-1),ncol=2)
  Y = matrix(rnorm(nn*2, mean=+1),ncol=2)

  vdist[i] = wasserstein(X, Y, p=2)$distance
  if (i%%10 == 0){
    print(paste0("iteration ",i,"/", niter," complete."))
  }
}

# Visualize
opar <- par(no.readonly=TRUE)
hist(vdist, main="Monte Carlo Simulation")
abline(v=sqrt(8), lwd=2, col="red")
par(opar)

## End(Not run)
```

Index