Package 'T4transport'

May 29, 2025

Type Package

Title Tools for Computational Optimal Transport

Version 0.1.3

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 CVXR, Rcpp (>= 1.0.5), Rdpack, lpSolve, stats, utils

LinkingTo Rcpp, RcppArmadillo, BH

Encoding UTF-8

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

RoxygenNote 7.3.2

RdMacros Rdpack

Suggests ggplot2

Depends R (>= 2.10)

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 2025-05-29 21:00:02 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)
```

---

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 $N(\mu_i, \Sigma_i^2)$ 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

```
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 (2024). "On the Wasserstein Median of Probability Measures." *Journal of Computational and Graphical Statistics*, 1–25. ISSN 1061-8600, 1537-2715, doi:10.1080/10618600.2024.2374580.

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 $\mathcal{N}(\mu_i, \sigma_i^2)$ 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 (2024). "On the Wasserstein Median of Probability Measures." *Journal of Computational and Graphical Statistics*, 1–25. ISSN 1061-8600, 1537-2715, doi:10.1080/10618600.2024.2374580.

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)
```

`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 of 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)
```

`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)
```

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)
```

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")
points(myatoms[[2]], col="gray90", pch=19, cex=0.5)
points(myatoms[[3]], col="gray90", pch=19, cex=0.5)
points(myatoms[[4]], col="gray90", pch=19, cex=0.5)

#  plot the barycenter
points(fsbary$support, col="red", cex=1.5)
par(opar)
```

`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)
```

`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 `wassersteinD()` 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