

Package: T4transport (via r-universe)

October 2, 2026

Type Package

Title Tools for Computational Optimal Transport

Version 0.2.0.9002

Description Computational optimal transport for weighted finite measures, Gaussian distributions, and finite dissimilarity spaces. Provides exact and regularized transport, sliced and alignment-based comparisons, Gromov-Wasserstein calculations, barycenters, medians, and interpolation. Explicit representations and result inspection distinguish mathematical targets, numerical termination, and approximation. See Peyre and Cuturi (2019) <[doi:10.1561/22000000073](https://doi.org/10.1561/22000000073)> for a general exposition.

License MIT + file LICENSE

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

LinkingTo Rcpp, RcppArmadillo

Encoding UTF-8

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

RdMacros Rdpack

Depends R (>= 4.1.0)

Suggests knitr, rmarkdown, ggplot2, mlbench, testthat (>= 3.0.0)

SystemRequirements C++20

VignetteBuilder knitr

Config/roxygen2/version 8.1.0

Config/testthat/edition 3

Repository <https://kisungyou.r-universe.dev>

Date/Publication 2026-10-02 15:35:31 UTC

RemoteUrl <https://github.com/kisungyou/t4transport>

RemoteRef HEAD

RemoteSha 1d56c5a0a490f15f9e679d96d17c093519b4e883

Contents

digit3	3
digits	4
ecdfbary	5
ecdfmed	6
fbary14C	8
fbary15B	10
fiedler	12
gaussbary1d	14
gaussbarypd	15
gaussmed1d	17
gaussmedpd	18
gaussvis2d	20
gwbary	22
gwdist	23
histbary	26
histbary14C	27
histbary15B	29
histdist	31
histinterp	32
histmed	34
imagebary	36
imagebary14C	38
imagebary15B	40
imagedist	42
imageinterp	43
imagemed	44
img2measure	45
ipot	47
ot_cost	49
ot_distance	49
ot_gaussian	50
ot_gaussian_distance	51
ot_gaussian_summary	52
ot_image	53
ot_inspect	54
ot_measure	55
ot_summary	56
pwbary	57
pwdist	60
quantile.t4transport_cdf	62
rbary23L	63
rbaryGD	64
rmedIRLS	67
rmedWB	69
sinkhorn	72
swdist	74

<i>digit3</i>	3
wassboot	75
wasserstein	77
Index	80

digit3	<i>MNIST Images of Digit 3</i>
--------	--------------------------------

Description

`digit3` contains 2000 images from the MNIST dataset of digit 3. Each element of the list is an image represented as an (28×28) matrix that sums to 1. Normalization makes intensities into probability masses and discards the original total intensity of each image.

Usage

```
data(digit3)
```

Format

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

Source

MNIST Database of Handwritten Digits, by Yann LeCun, Corinna Cortes, and Christopher J.C. Burges, <http://yann.lecun.com/exdb/mnist/>. The original package source does not preserve this subset's MNIST row indices, train/test designation, random seed, or exact preparation code. See the installed 'DATA_PROVENANCE' for source-artifact checksums and the limits of the provenance and dataset-specific licensing evidence.

References

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 2278–2324. doi:10.1109/5.726791.

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	<i>MNIST Images of All Digits</i>
--------	-----------------------------------

Description

digits contains 5000 images from the 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. Normalization makes intensities into probability masses and discards the original total intensity of each image.

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.

Source

MNIST Database of Handwritten Digits, by Yann LeCun, Corinna Cortes, and Christopher J.C. Burges, <http://yann.lecun.com/exdb/mnist/>. The retained preparation fragment uses `dslabs::read_mnist()`, samples 500 training images per class without replacement, reshapes each row with `matrix(pixel_vector, nrow = 28)[, 28:1]`, and normalizes total mass. Its original random seed, selected MNIST row indices, and dslabs version were not recorded. See the installed 'DATA_PROVENANCE' for checksums, transformations, and limits of the dataset-specific licensing evidence.

References

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 2278–2324. doi:10.1109/5.726791.

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 $i = 1, \dots, N$, compute the Wasserstein barycenter of order 2. This is obtained by taking a weighted average on a set of corresponding quantile functions on their exact common refinement. Ties and unequal empirical sample sizes retain their probability masses.

Usage

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

Arguments

ecdfs	a nonempty list of base "ecdf" objects from [stats::ecdf()] or weighted CDFs returned by these summary functions.
weights	outer weights across input distributions; if NULL (default), uniform weight is set. Otherwise, it should be a length- N vector of nonnegative weights.
...	extra parameters including abstol absolute numerical tolerance (default: 1e-8). reltol relative numerical tolerance (default: 1e-8). maxiter maximum number of median updates (default: 1000). The barycenter is computed directly.

Value

A callable, right-continuous weighted CDF of class "t4transport_cdf", which also inherits from "stepfun". Its "t4transport_measure" attribute contains exact support locations and masses; "t4transport_fit" contains the objective, accepted history, and numerical diagnostics. It can be supplied to subsequent calls to ecdfbary() or ecdfmed(). No artificial repeated sample is constructed.

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 $i = 1, \dots, N$, compute the Wasserstein median. This is obtained by a collision-aware modified Weiszfeld algorithm in exact finite quantile coordinates. The target is a sum of Wasserstein-2 distances, not coordinatewise quantile medians. A warning is issued if the stationarity criterion is not met; details remain in the returned fit attribute.

Usage

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

Arguments

<code>ecdfs</code>	a nonempty list of base "ecdf" objects from <code>[stats::ecdf()]</code> or weighted CDFs returned by these summary functions.
<code>weights</code>	outer weights across input distributions; if <code>NULL</code> (default), uniform weight is set. Otherwise, it should be a length- N vector of nonnegative weights.
<code>...</code>	extra parameters including abstol absolute numerical tolerance (default: 1e-8). reltol relative numerical tolerance (default: 1e-8). maxiter maximum number of median updates (default: 1000). The barycenter is computed directly.

Value

A callable, right-continuous weighted CDF of class "t4transport_cdf", which also inherits from "stepfun". Its "t4transport_measure" attribute contains exact support locations and masses; "t4transport_fit" contains the objective, accepted history, and numerical diagnostics. It can be supplied to subsequent calls to `ecdfbary()` or `ecdfmed()`. No artificial repeated sample is constructed.

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 $\mu_1, \mu_2, \dots, \mu_K$ 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. The compatibility implementation is described in Details.

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.

Details

Since version 0.2.0, the historical entry points use the shared objective-checked mirror-descent solver with log-domain Sinkhorn inner solves. Their names are retained for compatibility, not as a claim that the original accelerated or Bregman-projection iteration is still used. The optimized cost is distance raised to p, plus $\lambda * \sum(\text{plan} * (\log(\text{plan}) - 1))$. Locations are fixed and candidate probability masses vary. Inspect status and the reported convex objective bound; exhausting the iteration budget is not success. Additional controls are `rtol`, `step0`, `max_backtrack`, `inner_maxiter`, `inner_tol`, and `return_result`. Initial masses must be strictly positive.

Value

A numeric `t4transport_mass` vector with a diagnostics attribute and a compatible barplot method. Use `return_result = TRUE` to return the full fit instead.

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 = 20
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 $\mu_1, \mu_2, \dots, \mu_K$ 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. The historical method used iterative Bregman projections. The current compatibility implementation is described in Details.

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.

Details

Since version 0.2.0, the historical entry points use the shared objective-checked mirror-descent solver with log-domain Sinkhorn inner solves. Their names are retained for compatibility, not as a claim that the original accelerated or Bregman-projection iteration is still used. The optimized cost is distance raised to p , plus $\lambda \sum (\text{plan} * (\log(\text{plan}) - 1))$. Locations are fixed and candidate probability masses vary. Inspect status and the reported convex objective bound; exhausting the iteration budget is not success. Additional controls are `rtol`, `step0`, `max_backtrack`, `inner_maxiter`, `inner_tol`, and `return_result`. Initial masses must be strictly positive.

Value

A numeric `t4transport_mass` vector with a diagnostics attribute and a compatible barplot method. Use `return_result = TRUE` to return the full fit instead.

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 = 20
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)
```

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

<code>X</code>	An $(N \times P)$ matrix of row observations.
<code>normalize</code>	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.

Details

This graph summary is not a transport distance. The eigenvector sign is fixed by making its largest absolute component positive; repeated eigenvalues can still give different bases on different platforms. At least two points and a positive median pairwise distance are required.

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 $i = 1, \dots, n$, compute the Wasserstein-2 barycenter exactly: average the means and standard deviations separately, then square the resulting standard deviation. The iteration controls are retained and validated for compatibility.

Usage

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

Arguments

means	a length- n vector of mean parameters.
vars	a length- n vector of finite nonnegative variances. Zero denotes a degenerate Gaussian point mass.
weights	outer weights across Gaussian distributions; if NULL (default), uniform weight is set. Otherwise, it should be a length- n vector of nonnegative weights.
...	extra parameters including abstol absolute numerical tolerance (default: 1e-8). reitol relative numerical tolerance (default: 1e-8). maxiter maximum number of iterations (default: 496).

Value

A named list preserving the mean and variance fields below, with objective, accepted history, status, stopping reason, iteration count, and numerical diagnostics. Non-success produces a warning.

mean mean of the estimated barycenter distribution.

var variance of the estimated barycenter distribution.

References

There are no references for Rd macro \insertAllCites on this help page.

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 $\mathbb{R}^p$ **Description**

Given a collection of p -dimensional Gaussian distributions $N(\mu_i, \Sigma_i)$ for $i = 1, \dots, n$, 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 of finite symmetric strictly positive-definite covariance matrices. No covariance projection or ridge is applied.

weights outer weights across Gaussian distributions; if NULL (default), uniform weight is set. Otherwise, it should be a length- n vector of nonnegative weights.

... extra parameters including

- abstol** absolute numerical tolerance (default: 1e-8).
- reltol** relative numerical tolerance (default: 1e-8).
- maxiter** maximum number of iterations (default: 496).

Value

A named list preserving the mean and variance fields below, with objective, accepted history, status, stopping reason, iteration count, and numerical diagnostics. Non-success produces a warning.

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](https://doi.org/10.1016/j.jmaa.2016.04.045).

See Also

[T4transport::gaussbaryld()] 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 $i = 1, \dots, n$, compute the median under Wasserstein-2 distance within the Gaussian family. This is a collision-aware Euclidean geometric median of the pairs (mean, standard deviation), not separate coordinatewise medians.

Usage

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

Arguments

<code>means</code>	a length- n vector of mean parameters.
<code>vars</code>	a length- n vector of finite nonnegative variances. Zero denotes a degenerate Gaussian point mass.
<code>weights</code>	outer weights across Gaussian distributions; if NULL (default), uniform weight is set. Otherwise, it should be a length- n vector of nonnegative weights.
<code>...</code>	extra parameters including abstol absolute numerical tolerance (default: 1e-8). reltol relative numerical tolerance (default: 1e-8). maxiter maximum number of iterations (default: 496).

Value

A named list preserving the mean and variance fields below, with objective, accepted history, status, stopping reason, iteration count, and numerical diagnostics. Non-success produces a warning.

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)
```

Description

Given a collection of p -dimensional Gaussian distributions $N(\mu_i, \Sigma_i)$ for $i = 1, \dots, n$, compute the Wasserstein-2 median within the strictly SPD Gaussian family. General covariance fits use iteratively reweighted squared-distance subproblems with arithmetic mean updates and covariance barycenters. Objective and iterate tolerances are numerical stopping criteria; they do not certify a globally optimal median.

Usage

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

Arguments

<code>means</code>	an $(n \times p)$ matrix whose rows are mean vectors.
<code>vars</code>	a $(p \times p \times n)$ array of finite symmetric strictly positive-definite covariance matrices. No covariance projection or ridge is applied.
<code>weights</code>	outer weights across Gaussian distributions; if NULL (default), uniform weight is set. Otherwise, it should be a length- n vector of nonnegative weights.
<code>...</code>	extra parameters including abstol absolute numerical tolerance (default: 1e-8). reitol relative numerical tolerance (default: 1e-8). maxiter maximum number of iterations (default: 496). inner_maxiter covariance-subproblem iteration limit (default: 496). inner_abstol, inner_reitol absolute and relative covariance-subproblem tolerances (default: 1e-10).

Value

A named list preserving the mean and variance fields below, with objective, accepted history, status, stopping reason, iteration count, and numerical diagnostics. Non-success produces a warning.

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

Draw a Bivariate Gaussian Covariance Ellipse

Description

This function places deterministic points along the contour of an ellipse represented by mean and variance parameters for a 2-dimensional Gaussian distribution to help ease manipulating visualiza-

tion of the specified distribution. For example, you can directly use a basic `plot()` function directly for drawing.

Usage

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

Arguments

<code>mean</code>	a length-2 vector for mean parameter.
<code>var</code>	a (2×2) matrix for covariance parameter.
<code>n</code>	Integer number of contour points, at least 3 (default: 500).

Details

The contour has Mahalanobis radius one; it is not a random sample or an automatically calibrated confidence region. Covariance must be strictly positive definite, with no silent ridge or projection repair.

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 symmetric dissimilarity space representing the input collection under a local GW squared-loss criterion. Triangle inequalities are not imposed, so the returned object need not be a metric space. Global optimality is not certified.

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 <code>dist</code> representing the pairwise distances for each empirical measure.
marginals	marginal distributions for empirical measures; if <code>NULL</code> (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 <code>NULL</code> (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 GW comparison method: "mm", "pg", or "fw" (default); see gwdist. inner_maxiter, inner_tol GW comparison iteration limit and tolerance (defaults: 100, 1e-8). transport_maxiter exact transport iteration limit (default: 100000000; zero means no limit). init_dist optional initial symmetric zero-diagonal dissimilarity matrix. projection_maxiter, projection_tol, max_backtrack PG controls passed to gwdist.

Value

A named list containing

dist an object of class `dist` representing the attained dissimilarity barycenter; a metric is not guaranteed.

weight a length- M vector of fixed barycenter masses with all entries being $1/M$. Status, objective history, dissimilarity matrix, and pairwise diagnostics are also returned.

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(digit3)
data_D = vector("list", length=5)
data_W = vector("list", length=5)
for (i in 1:5){
  img_now = img2measure(digit3[[i]])
  data_D[[i]] = stats::dist(img_now$support)
  data_W[[i]] = as.vector(img_now$weight)
}

## COMPUTE
bary_dist <- gw_bary(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)
```

Description

Computes a local Gromov-Wasserstein (GW) comparison between finite spaces. Given two distance matrices D_X and D_Y along with their respective marginal distributions, the function solves the GW optimization problem approximately to obtain a squared-loss objective and an associated coupling. Global optimality is not certified.

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: Frank–Wolfe (FW) with quadratic line search, a full linear-oracle update (the historical MM label), and projected gradient (PG) with Dykstra Euclidean projection and backtracking. These are unregularized nonconvex iterations. Symmetry and zero-diagonal checks do not establish the triangle inequality. The objective is the full squared-loss sum, without a factor 1/2.

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 probability mass vector that sums to 1. If NULL (default), uniform weight is set.
wy	a length- N probability mass vector 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 one of "mm", "pg", or "fw" (default). inner_maxiter exact linear transport iteration limit (default: 100000000; zero means no limit). init_plan optional feasible starting coupling; default is the weighted index coupling. projection_maxiter, projection_tol Dykstra projection controls for PG (defaults: 10000, 1e-10). max_backtrack maximum PG step halvings (default: 25).

Details

Squared-loss evaluations use a fast matrix expansion, with direct nonnegative block sums where a roundoff bound indicates cancellation. This avoids silently reporting zero for nearly equal large dissimilarities. The fallback uses bounded working memory but may require quartic arithmetic in the worst case. Costs outside floating-point range fail explicitly.

Value

a named list containing

distance square root of the attained squared-loss objective, not a certified global GW distance.

plan an $(M \times N)$ nonnegative coupling matrix. The result also records objective, status, first-order gap, and accepted history.

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

Compute the finite 1D 2-Wasserstein barycenter of midpoint-discrete histograms, then return its projection onto the original histogram bins. All input histograms must have identical breaks. The unprojected finite measure is retained separately.

Usage

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

Arguments

<code>hists</code>	a length- N list of histograms ("histogram" objects) of same breaks.
<code>weights</code>	a weight for each histogram; if <code>NULL</code> (default), uniform weights are used. Otherwise, it should be a length- N vector of nonnegative weights.
<code>L</code>	retained for compatibility and validated as a positive integer. The finite quantile backend uses exact mass intervals, so this argument no longer controls an approximation grid.

Details

Inputs are discrete measures at bin midpoints, using normalized counts (or the canonical mass field of an estimated histogram). The unprojected finite result is saved in `unprojected_measure`. For the legacy histogram display it is binned onto the original breaks, using right-closed bins with the lowest endpoint included, and represented at bin midpoints. This projection generally changes the measure;

it is not an exact unrestricted Wasserstein summary or geodesic. Downstream histogram functions consistently use the displayed midpoint measure, not the separate unprojected result. Break values must agree exactly across inputs; no grid snapping is done. When an estimated histogram is reused, its previous computational metadata is discarded. Returned diagnostics describe the current calculation. Returned mass and counts are the same probabilities; density equals mass divided by bin width. Counts are not rounded observations, and `effective_sample_size` is NA because an estimated measure has no implied sample size. The `objective/history` and `solver` status refer to the unprojected summary.

Value

a "histogram" display of the barycenter, with the unprojected estimate and projection metadata.

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)
```

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. Break values must agree exactly across inputs; no grid snapping is done.

Usage

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

Arguments

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

Details

Each input is the discrete measure at its bin midpoints with normalized bin counts, including unequal-width bins. The entropy coefficient `lambda` multiplies the sum of transport-plan entries times their logarithms; the cost is midpoint distance raised to `p`. The finite regularization coefficient changes the objective from the unregularized barycenter target. The shared solver now uses log-domain Sinkhorn gradients and objective-checked mirror descent; historical names remain compatibility entry points. Returned mass and counts are probability masses, and density is mass divided by bin width. No sample-size-dependent rounding is performed; `effective_sample_size` is NA. Status, accepted objective history, and inner diagnostics are available in `fit`. The full objective uses $\log(\text{plan}) - 1$; this differs from $\log(\text{plan})$ by a constant and has the same minimizer.

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. Break values must agree exactly across inputs; no grid snapping is done.

Usage

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

Arguments

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

Details

Each input is the discrete measure at its bin midpoints with normalized bin counts, including unequal-width bins. The entropy coefficient `lambda` multiplies the sum of transport-plan entries times their logarithms; the cost is midpoint distance raised to `p`. The finite regularization coefficient changes the objective from the unregularized barycenter target. The shared solver now uses log-domain Sinkhorn gradients and objective-checked mirror descent; historical names remain compatibility entry points. Returned mass and counts are probability masses, and density is mass divided by bin width. No sample-size-dependent rounding is performed; `effective_sample_size` is NA. Status, accepted objective history, and inner diagnostics are available in `fit`. The full objective uses $\log(\text{plan}) - 1$; this differs from $\log(\text{plan})$ by a constant and has the same minimizer.

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](https://doi.org/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 or canonical estimated bin masses. Uses the exact 1D monotone OT algorithm, not LP nor entropic regularization. Break values must agree exactly; to compare different grids, convert each histogram with `as_ot_measure()` and use `ot_distance()`.

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

Compute exact finite displacement interpolation between midpoint-discrete histograms with identical breaks, then project each result onto the original histogram bins. The unprojected finite measures follow the 2-Wasserstein geodesic; the projected histogram displays generally do not.

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. Endpoints preserve the corresponding input probability measure.
L	retained for compatibility and validated as a positive integer. The finite quantile backend uses exact mass intervals, so this argument no longer controls an approximation grid.

Details

Inputs are discrete measures at bin midpoints, using normalized counts (or the canonical mass field of an estimated histogram). The unprojected finite result is saved in `unprojected_measure`. For the legacy histogram display it is binned onto the original breaks, using right-closed bins with the lowest endpoint included, and represented at bin midpoints. This projection generally changes the measure; it is not an exact unrestricted Wasserstein summary or geodesic. Downstream histogram functions consistently use the displayed midpoint measure, not the separate unprojected result. Break values must agree exactly across inputs; no grid snapping is done. When an estimated histogram is reused, its previous computational metadata is discarded. Returned diagnostics describe the current calculation. Returned mass and counts are the same probabilities; density equals mass divided by bin width. Counts are not rounded observations, and `effective_sample_size` is NA because an estimated measure has no implied sample size. The objective/history and solver status refer to the unprojected summary.

Value

If `length(t) == 1`, a single "histogram" object representing the projected interpolation at time `t`, including the unprojected measure. 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

Compute a 1D Wasserstein geometric median of midpoint-discrete histograms with common breaks, using a modified Weiszfeld algorithm on exact finite quantile intervals. Return its projection onto the original histogram bins and retain the unprojected finite estimate separately.

Usage

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

Arguments

<code>hists</code>	a length- N list of histograms ("histogram" objects) with identical breaks.
<code>weights</code>	a weight for each histogram; if <code>NULL</code> (default), uniform weights are used. Otherwise, it should be a length- N vector of nonnegative weights.
<code>L</code>	retained for compatibility and validated as a positive integer. The finite quantile backend uses exact mass intervals, so this argument no longer controls an approximation grid.

```

...      extra parameters including
          abstol absolute solver tolerance (default: 1e-8).
          reltol relative solver tolerance (default: 1e-8).
          maxiter maximum number of iterations (default: 496).
          print.progress logical; whether to show current iteration (default: FALSE).

```

Details

Inputs are discrete measures at bin midpoints, using normalized counts (or the canonical mass field of an estimated histogram). The unprojected finite result is saved in `unprojected_measure`. For the legacy histogram display it is binned onto the original breaks, using right-closed bins with the lowest endpoint included, and represented at bin midpoints. This projection generally changes the measure; it is not an exact unrestricted Wasserstein summary or geodesic. Downstream histogram functions consistently use the displayed midpoint measure, not the separate unprojected result. Break values must agree exactly across inputs; no grid snapping is done. When an estimated histogram is reused, its previous computational metadata is discarded. Returned diagnostics describe the current calculation. Returned mass and counts are the same probabilities; density equals mass divided by bin width. Counts are not rounded observations, and `effective_sample_size` is NA because an estimated measure has no implied sample size. The objective/history and solver status refer to the unprojected summary.

Value

a "histogram" display of the median, with the unprojected estimate, solver diagnostics, and projection metadata.

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^* from multiple input images $X_1, \dots, X_N$. 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

<code>images</code>	a length- N list of same-size image matrices of size $(m \times n)$.
<code>p</code>	an exponent for the order of the distance (default: 2). Currently, only $p=2$ is supported (squared ground distance cost).
<code>weights</code>	a weight of each image; if <code>NULL</code> (default), uniform weight is set. Otherwise, it should be a length- N vector of nonnegative weights.
<code>C</code>	an optional $(mn \times mn)$ ground cost matrix. If <code>NULL</code> (default), the squared Euclidean grid cost is used. Providing <code>C</code> allows using alternative ground costs (e.g., geodesic distances on a manifold discretization).
<code>...</code>	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 candidate mass floor before normalization (default: 1e-15). Inputs are unchanged. With zero floor, initialization must be strictly positive.

smooth optional mixing weight toward uniform distribution after each update, used to perturb the candidate toward uniform mass (default: 0). Set to 0 to disable.

clip ℓ_∞ clipping threshold for the subgradient to stabilize exponentials in the KL update (default: Inf, disabled).

max_backtrack maximum number of backtracking halvings of the stepsize when an OT probe would increase the full objective (default: 8).

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

inner_maxiter exact-solver optimization-pivot budget, excluding initialization (default: 100000000); zero means unlimited.

return_result return a list with estimate, objective/history, status, inner diagnostics, controls, and candidate perturbation totals (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 optional candidate smoothing and backtracking against the full objective. Input masses are never floored. Inner-solver failures return a failed result; they do not trigger an automatic change to the input distributions.

Value

An $(m \times n)$ matrix with a diagnostics attribute, or a diagnostic result list when `return_result = TRUE`. Non-successful matrix returns issue a warning. Small iterate change does not certify global optimality of this nonsmooth numerical optimization.

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 $X_1, X_2, \dots, X_N$. 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

<code>images</code>	a length- N list of same-size image matrices of size $(m \times n)$.
<code>p</code>	an exponent for the order of the distance (default: 2).
<code>weights</code>	a weight of each image; if NULL (default), uniform weight is set. Otherwise, it should be a length- N vector of nonnegative weights.
<code>lambda</code>	a regularization parameter; if NULL (default), a paper's suggestion would be taken, or it should be a strictly positive real number.
<code>...</code>	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 compatibility control; only 1 is supported. print.progress a logical to show final solver diagnostics (default: FALSE).

Details

Since version 0.2.0 the historical names call the shared objective-checked mirror-descent solver with log-domain Sinkhorn gradients, as documented in [fbary14C](#). The entropy coefficient `lambda` multiplies $\text{sum}(\text{plan} * (\log(\text{plan}) - 1))$. Pixel columns run from $x=0$ to $x=1$ and rows from $y=1$ to $y=0$; a singleton axis is fixed at its first coordinate. Pixel intensities are probability masses after normalization. Additional controls from `fbary14C` are accepted, using `init.image` instead of `init.vec`.

Value

A matrix of image probability masses with a diagnostics attribute; `return_result = TRUE` returns a rich fit and 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 $X_1, X_2, \dots, X_N$. 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 strictly positive 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 compatibility control; only 1 is supported.
	print.progress a logical to show final solver diagnostics (default: FALSE).

Details

Since version 0.2.0 the historical names call the shared objective-checked mirror-descent solver with log-domain Sinkhorn gradients, as documented in [fbary14C](#). The entropy coefficient `lambda` multiplies $\sum(\text{plan} * (\log(\text{plan}) - 1))$. Pixel columns run from $x=0$ to $x=1$ and rows from $y=1$ to $y=0$; a singleton axis is fixed at its first coordinate. Pixel intensities are probability masses after normalization. Additional controls from [fbary14C](#) are accepted, using `init.image` instead of `init.vec`.

Value

A matrix of image probability masses with a diagnostics attribute; `return_result = TRUE` returns a rich fit and 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](https://doi.org/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
datsmall = list(matrix(c(1, 0, 0, 1), 2, 2))
outsmall = imagebary15B(datsmall, lambda=0.5, maxiter=3)
attr(outsmall, 'diagnostics')$status

## 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)))
```

Description

Given two grayscale images represented as numeric matrices of identical size, compute interpolated images along a 2-Wasserstein geodesic connecting them, then projects it onto the original grid. 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: 0). A positive value changes the inputs; its mass perturbation is reported.
	inner_maxiter exact solver pivot budget (default: 100000000).
	project only nearest is supported.
	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. The unprojected finite measure is retained in an attribute; the displayed matrix generally is not an exact Wasserstein geodesic.

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

Median on a Prescribed Image Grid

Description

Minimizes the weighted sum of W_2 distances over probability masses on the common image grid, using exact OT values and objective-checked mass updates. The mass-space median objective is nonconvex. A local stopping criterion is not a global-optimality certificate; a dominating input can be certified separately by the triangle inequality for the default Euclidean geometry.

Usage

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

Arguments

<code>images</code>	A nonempty list of nonnegative finite image matrices of the same size.
<code>weights</code>	Nonnegative outer weights, normalized to sum to one.
<code>C</code>	Optional finite nonnegative squared-cost matrix on the pixel grid. An arbitrary supplied matrix defines a sum of rooted OT costs, with no metric or median robustness guarantee asserted.
<code>...</code>	Named controls: <code>maxiter</code> (30), <code>abstol</code> (1e-6), <code>rtol</code> (1e-8), <code>init.image</code> (uniform positive mass), <code>step0</code> (1), <code>max_backtrack</code> (30), <code>inner_maxiter</code> (10000), <code>inner_tol</code> (1e-10), <code>print.progress</code> (FALSE), and <code>return_result</code> (FALSE). Unsupported controls are rejected.

Details

Since version 0.2.0, the old denominator-floor IRLS and its nested `bary.*delta` controls have been replaced by the shared fixed-support median solver. Input zero masses are retained exactly. An uncertified stationary point, ambiguous zero-distance collisions, and inner solver failures are reported; no failed input is discarded or downweighted. Pixel columns span $[0,1]$ and rows run from $y=1$ to $y=0$, as in [imagedist](#).

Value

An image probability-mass matrix with a diagnostics attribute, or a full fit when `return_result = TRUE`.

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)
```

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	A nonempty nonnegative numeric image 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.

Details

This legacy converter uses pixel units and base `image()`'s row-as-x orientation. It differs from the endpoint grid of `imagedist` and the pixel-centre grid of `ot_image`. Choose a coordinate convention explicitly when comparing measures. Thresholding reports removed mass.

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 KL proximal formulation. Log scaling reduces kernel underflow but does not remove all floating-point limitations.

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.

The returned value is the rooted transport cost of an approximate plan. Small changes in the plan and small marginal residuals are stopping criteria, not a certificate of optimality for the unregularized problem. The parameter `lambda` controls proximal steps and is not an entropy penalty in the reported objective.

Usage

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

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

Arguments

<code>X</code>	an $(M \times P)$ matrix of row observations.
<code>Y</code>	an $(N \times P)$ matrix of row observations.
<code>p</code>	an exponent for the order of the distance (default: 2).
<code>wx</code>	a length- M probability mass vector that sums to 1. If <code>NULL</code> (default), uniform weight is set.

wy	a length- N probability mass vector that sums to 1. If NULL (default), uniform weight is set.
lambda	a regularization parameter (default: 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 pth root of the approximate plan's transport cost; not a certified Wasserstein distance

plan an $(M \times N)$ approximate transport plan. Status, marginal residuals, and an iteration history are also returned.

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)

```

ot_cost	<i>Declare supplied transport costs or ground distances</i>
---------	---

Description

Declare supplied transport costs or ground distances

Usage

```
ot_cost(values, type = c("cost", "distance"), metric = FALSE)
```

Arguments

values	A finite nonnegative cross-matrix.
type	Whether entries are already powered costs or ground distances.
metric	Whether the caller declares that values derive from a metric. This is a declaration, not a verified triangle inequality.

Value

A ‘t4transport_cost’ object. Distance entries are powered exactly once by the order supplied to ‘ot_distance’; cost entries are used unchanged.

ot_distance	<i>Compare finite probability measures</i>
-------------	--

Description

Exact methods target the unregularized transport problem. Sinkhorn uses ‘cost + epsilon * sum(plan * (log(plan) - 1))’; the transport component and full regularized objective are returned separately. A rooted regularized transport component is not an exact Wasserstein distance or a metric. The network-simplex backend normalizes active costs internally and returns objectives and potentials in the supplied cost units. Its normalized dual checks use the maximum active cost (one when all active costs are zero) and total mass as scales; these checks do not promise relative accuracy with respect to an arbitrarily small optimum within a large cost matrix.

Usage

```
ot_distance(
  x,
  y,
  p = 2,
  method = c("auto", "exact", "sinkhorn", "1d"),
  epsilon = 0.1,
  cost = NULL,
  control = list(),
  return_plan = FALSE
)
```

Arguments

x, y	Finite measures or inputs convertible by ‘as_ot_measure()’.
p	Wasserstein order, at least one.
method	‘auto’ uses monotone transport for scalar supports and network simplex otherwise; ‘exact’, ‘sinkhorn’, and ‘1d’ select explicit backends.
epsilon	Positive entropy coefficient for Sinkhorn.
cost	Optional explicit ‘ot_cost()’ object.
control	Named controls. Exact: ‘maxiter’. Sinkhorn: ‘maxiter’, ‘atol’. The finite 1D backend has no iteration controls.
return_plan	Whether to retain the plan in the R result. This does not promise a matrix-free solver or avoid internal dense allocations.

Value

A ‘t4transport_comparison’ with objective components, method, status, residuals, and optional plan. Non-success produces a warning and an explicit failed/limited result, with no claimed exact Wasserstein distance.

Examples

```
x <- ot_measure(c(0, 2), c(1, 3))
y <- ot_measure(c(0, 3), c(1, 1))
ot_distance(x, y)
ot_distance(x, y, method = "sinkhorn", epsilon = 0.5)
```

ot_gaussian

Parametric Gaussian probability measures

Description

Keep Gaussian parameters separate from finite samples. Univariate variance zero is supported as a degenerate point mass. Covariances in dimensions greater than one must be strictly positive definite. No simulation, covariance ridge, or PSD projection is performed.

Usage

```
ot_gaussian(mean, covariance)
```

Arguments

mean	A nonempty finite real mean vector.
covariance	A covariance matrix matching the mean dimension; a scalar variance is also accepted for a univariate Gaussian.

Value

A ‘t4transport_gaussian’ object containing mean, covariance (‘var’), dimension, and its covariance-domain declaration.

Examples

```
x <- ot_gaussian(0, 1)
y <- ot_gaussian(3, 4)
ot_gaussian_distance(x, y)
```

ot_gaussian_distance *Exact Wasserstein-2 distance between Gaussian parameter objects*

Description

The covariance calculation uses a matrix-factor Procrustes identity to avoid subtracting nearly equal covariance traces. The answer concerns the Gaussian distributions themselves, not an empirical sample from them.

Usage

```
ot_gaussian_distance(x, y)
```

Arguments

x, y	Objects returned by [ot_gaussian()].
------	--------------------------------------

Value

A ‘t4transport_comparison’ with Wasserstein distance and squared cost. No finite transport plan is constructed.

ot_gaussian_summary *Summaries of Gaussian parameter objects*

Description

Compute a Wasserstein-2 barycenter or a median within the Gaussian family using the existing parameter-based algorithms. The barycenter minimizes the weighted sum of squared distances; the median minimizes weighted distances. No finite sampling, covariance ridge, or covariance projection is performed.

Usage

```
ot_gaussian_summary(
  measures,
  target = c("barycenter", "median"),
  weights = NULL,
  control = list()
)
```

Arguments

measures	A nonempty list of [ot_gaussian()] objects of the same dimension. One-dimensional variances may be zero; covariance matrices in higher dimensions must be strictly positive definite.
target	Either "barycenter" or "median".
weights	Nonnegative weights across Gaussian distributions, normalized separately. NULL assigns equal weights. Zero-weight inputs are validated but do not enter the optimization.
control	A named list. Outer controls are 'maxiter = 496', 'atol = 1e-8', and 'rtol = 1e-8'. Only multidimensional medians also accept 'inner_maxiter = 496', 'inner_atol = 1e-10', and 'inner_rtol = 1e-10'. These map to the legacy solvers' iteration and absolute/relative tolerance controls. Controls not used by an analytic branch are recorded in 'settings\$controls_unused'; unknown or inapplicable controls are errors.

Details

One-dimensional barycenters average means and standard deviations exactly. One-dimensional medians use the joint mean/standard-deviation geometric median. Multivariate calls use [gauss-barypd()] or [gaussmedpd()], including their diagonal and common-covariance reductions. General multivariate median termination is a numerical stopping result, not a global-optimality claim.

Value

A 't4transport_gaussian_fit' containing 'estimate' (a Gaussian parameter object), legacy 'mean' and 'var' fields, the objective, accepted history, status, stopping reason, iteration count, and diagnostics.

‘weights’ records normalized outer weights; ‘control’ records public controls and ‘backend_control’ records their legacy names. ‘settings’ describes the selected algorithm branch and unused controls. Non-success preserves the underlying solver’s warning and available estimate.

See Also

[ot_inspect()], [ot_gaussian_distance()], [gaussbary1d()], [gaussmed1d()], [gaussbarypd()], [gaussmedpd()]

Examples

```
measures <- list(ot_gaussian(0, 1), ot_gaussian(4, 9))
fit <- ot_gaussian_summary(measures, weights = c(1, 3))
fit$estimate
ot_gaussian_distance(measures[[1]], fit$estimate)
```

ot_image

Explicit image and density conversions

Description

Images use row-major order: x increases to the right and y increases upwards. Intensities are treated as pixel masses and normalized, discarding total intensity. Density grids use trapezoidal quadrature weights at the grid nodes; this is a finite approximation to the supplied curve, not a continuous density.

Usage

```
ot_image(image, x = NULL, y = NULL)

ot_density(grid, density)
```

Arguments

image	A nonempty nonnegative numeric matrix.
x, y	Optional pixel-center coordinates. x must increase by column and y must decrease by row. Values encode pixel mass, including on nonuniform grids.
grid	A strictly increasing finite numeric vector of at least two nodes.
density	Nonnegative finite density heights at the nodes.

Value

A finite ‘t4transport_measure’ recording conversion metadata.

Examples

```
ot_image(matrix(c(1, 0, 0, 2, 0, 1), 2, 3))
ot_density(seq(-3, 3, length.out = 21), dnorm(seq(-3, 3, length.out = 21)))
```

ot_inspect

*Inspect a transport computation without recomputing it***Description**

A common reporting view of the package's transport comparisons, barycenters, medians and interpolation results. Native results retain their existing classes, values and attributes. The report reads saved information only; it does not compute an objective, repair a candidate, or infer a certificate.

Usage

```
ot_inspect(x)
```

Arguments

x An individual result produced by a supported T4transport computation. For interpolation at several times, use `'lapply(results, ot_inspect)'`.

Details

The supported families are finite exact and regularized transport, sliced transport, Procrustes transport, Gromov–Wasserstein, and Gaussian comparisons and summaries, including their ECDF, histogram and image interfaces. Constructors, converters, `'fiedler()'`, `'gaussvis2d()'` and `'wassboot()'` are not computational-report inputs. Older saved results without producer metadata and arbitrary lists are not guessed from their field names.

A successful status retains its original stopping reason and scope. It need not certify a global optimum, a metric, or the accuracy of projection or sphere integration. Histogram diagnostics can concern the unprojected estimate, rather than the displayed midpoint measure. Entropy in the final objective, KL proximal steps, median smoothing and display projection are identified separately. An unavailable scalar is typed `'NA'`; an inapplicable component is empty. Existing `'NA'`, infinite residuals and failed statuses remain visible. Check norms and tolerances are left unavailable when the saved result does not identify them.

Value

A `'t4transport_report'` list with `'problem'` (family, target, geometry, feasible set, objective convention and regularization), named numeric `'values'`, `'termination'` (status, reason, convergence flag, iterations and scope), a `'checks'` data frame (quantity, value, norm, tolerance and scope), and lists `'approximation'`, `'history'` and `'settings'`. `'settings'` includes the producer and available method, controls, initialization and replay information. Histories retain the native saved format. No plan or estimated measure is copied into the report; inspect the native result for those.

Examples

```
result <- ot_distance(ot_measure(c(0, 2)), ot_measure(c(1, 3)))
ot_inspect(result)
fit <- ot_summary(lapply(c(0, 0, 3), ot_measure), target = "median")
summary(ot_inspect(fit))
```

Description

Represent atoms as rows of a numeric matrix and keep within-measure masses separate from weights across distributions. Nonnegative masses are normalized with overflow protection; zero masses and duplicate rows retain their indices.

Usage

```
ot_measure(
  support,
  mass = NULL,
  representation = "empirical",
  metadata = list()
)

as_ot_measure(x, ...)

ot_collection(measures, weights = NULL, labels = names(measures))
```

Arguments

support	A finite numeric vector or matrix; rows are atoms.
mass	Nonnegative masses, or NULL for uniform masses.
representation	A descriptive representation label.
metadata	A named list describing units and conversion choices.
x	An object to convert, or an object to print/subset.
...	Further arguments passed to the finite-measure constructor.
measures	A nonempty list of finite measures or convertible inputs.
weights	Weights across measures; NULL means equal importance per measure.
labels	Optional unique labels for the measures.

Value

‘ot_measure()’ and ‘as_ot_measure()’ return a ‘t4transport_measure’; ‘ot_collection()’ returns a ‘t4transport_collection’ with separately stored measures and outer weights.

Examples

```
mu <- ot_measure(c(0, 2), c(1, 3))
nu <- as_ot_measure(stats::ecdf(c(0, 0, 4)))
ot_collection(list(mu, nu), weights = c(2, 1))
```

ot_summary

*Barycenters and medians of finite probability measures***Description**

The unregularized barycenter minimizes the weighted sum of squared W2 distances; the median minimizes the weighted sum of W2 distances. The feasible set is explicit. Fixed-location entropy barycenters minimize the full regularized objectives; regularized medians are not substituted for unregularized medians. Free-support fits keep candidate masses fixed.

Usage

```
ot_summary(
  measures,
  target = c("barycenter", "median"),
  weights = NULL,
  feasible = c("auto", "unrestricted_1d", "free_support_fixed_mass", "fixed_support"),
  method = c("auto", "quantile", "relocation", "subgradient"),
  support = NULL,
  mass = NULL,
  epsilon = NULL,
  control = list(),
  nstart = 1L,
  seed = NULL
)
```

Arguments

measures	A collection or list of convertible finite measures.
target	'barycenter' or 'median', both using inner Wasserstein order two.
weights	Outer weights, overriding a collection's weights if supplied.
feasible	'unrestricted_1d', 'free_support_fixed_mass', or 'fixed_support'. 'auto' selects unrestricted quantiles in one dimension, free support otherwise; an explicit support requires an explicit feasible set.
method	'quantile', 'relocation', 'subgradient', or automatic selection consistent with the feasible set. Unsupported combinations are errors.
support	Candidate locations: fixed grid for fixed support, or initial locations for free support. NULL lets the free-support initializer choose.
mass	Fixed candidate masses for free support. Initial variable masses on a fixed grid are instead supplied as 'control\$init_mass'.
epsilon	Optional positive entropy coefficient, only for a fixed-support barycenter. NULL means unregularized transport.
control	Named, method-specific numerical controls. Quantile methods use 'maxiter', 'atol', 'rtol'. Relocation adds 'stationarity_tol', 'alpha', 'smoothing', 'inner_maxiter', 'surrogate_steps', 'num_support', 'max_backtrack', 'print.progress'. Fixed support adds 'step0', 'max_backtrack', 'init_mass', 'inner_maxiter', 'inner_tol'.

nstart	Number of free-support starts. When support is supplied it is used for the first start; later starts sample pooled atoms with outer and within-measure probabilities. All initial supports and failed outcomes are recorded.
seed	Optional local RNG seed; the previous RNG state is restored.

Details

Fixed-support median stationarity alone is not an optimality certificate: all vertices are probed for descent, and an uncertified stationary point is returned with non-success status. Free-support relocation uses a residual from the selected optimal transport plans, which is not a certificate of a local minimum when optimal plans tie. A positive-cost stop with coincident active candidate atoms and nondegenerate input measures is marked ‘unresolved_degeneracy’ unless an independent certificate applies. Separate initial locations or multiple starts can help diagnose this case. Quantile summaries preserve an input measure when its quantile is selected; objectives and residuals describe the final stored finite measure, including any rounding introduced when quantile cells are converted to atom masses.

Value

A ‘t4transport_fit’ identifying the estimate, objective, feasible set, method, accepted history, numerical status, per-measure diagnostics and initialization. Multiple starts retain all outcomes and choose among successful runs using the same objective. If none succeeds the best finite failed run is returned with a non-success status.

Examples

```
measures <- lapply(c(0, 0, 0, 2, 8), ot_measure)
ot_summary(measures, target = "barycenter")
fit <- ot_summary(measures, target = "median")
summary(fit)
```

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.

This is a local block algorithm. It alternates orthogonal alignment, exact transport plans, and support relocation with uniform fixed candidate masses. It does not certify the global PW barycenter. Zero outer weights are excluded from optimization. Status, accepted objective history, and pairwise failures are returned, including when the outer iteration limit is reached.

Usage

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

Arguments

<code>atoms</code>	a length- K list of $(N_k \times P)$ atom matrices, a list of finite <code>[ot_measure()]</code> objects, or an <code>[ot_collection()]</code> .
<code>marginals</code>	marginal distributions for matrix inputs; if NULL (default), uniform masses are set. Otherwise supply a length- K list of nonnegative mass vectors, normalized separately. This must be NULL for measure objects or a collection, whose stored masses are authoritative.
<code>weights</code>	outer weights across measures, normalized internally. A non-NULL vector overrides collection weights. NULL inherits collection weights, or uses equal weights for a list.
<code>num_support</code>	the number of support points M for the PW barycenter (default: 100).
<code>...</code>	extra parameters including abstol stopping criterion for iterations (default: 1e-6). maxiter maximum number of outer iterations (default: 10). inner_maxiter, inner_tol PW comparison iteration limit and tolerance (defaults: 496, 1e-10). transport_maxiter exact transport iteration limit (default: 100000000; zero means no limit). init.support optional initial support matrix with <code>num_support</code> rows.

Value

a list with the following compatibility fields and optimization diagnostics:

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

weight a length- M vector of candidate masses 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 .

Alternating exact transport and orthogonal Procrustes updates give a local comparison; global minimization over alignments is not certified. The algorithm does not translate, center, or rescale the observations. Accepted objective history, initialization, controls, and termination status are returned. The default compares both initial alignments using successful exact solves only. A failed alternative is recorded in `initialization$attempts` without replacing an available valid start. If both starts fail, no plan or attained objective is returned.

Usage

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

Arguments

X	an $(M \times P)$ matrix of row observations or a finite measure from <code>[ot_measure()]</code> .
Y	an $(N \times P)$ matrix of row observations or a finite measure from <code>[ot_measure()]</code> .
wx	a length- M nonnegative mass vector, normalized internally. If <code>NULL</code> (default), matrix rows have uniform mass. For a measure object this must be <code>NULL</code> and its stored masses are used.
wy	a length- N nonnegative mass vector with the same rules as <code>wx</code> .
...	extra parameters including

maxiter maximum number of iterations (default: 496).
abstol absolute tolerance for the change of squared block objective (default: $1e-10$).
inner_maxiter iteration limit for each exact transport solve (default: 100000000; zero means no limit).
init_alignment optional orthogonal matrix; otherwise compare a weighted index-coupling initialization and identity.

Value

a named list containing

distance the computed PW distance value.

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

alignment the attained 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)

```

quantile.t4transport_cdf

Generalized inverse of a T4transport weighted CDF

Description

Generalized inverse of a T4transport weighted CDF

Usage

```

## S3 method for class 't4transport_cdf'
quantile(x, probs = seq(0, 1, 0.25), na.rm = FALSE, names = TRUE, ...)

```

Arguments

x	A weighted CDF returned by [ecdfbary()] or [ecdfmed()].
probs	Numeric probabilities in the closed unit interval.
na.rm	Accepted for generic compatibility; finite probabilities are required.
names	Whether to name results by their probabilities.
...	Only type = 1 is supported: the generalized inverse CDF.

Value

A numeric vector of inverse-CDF values. At zero and one the minimum and maximum positive-mass support locations are returned.

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](https://doi.org/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)
```

Description

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

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

is approximated by an iterative barycentric-projection update. The method is motivated by the formal first-order geometry of the 2-Wasserstein space according to Otto (2001), but is implemented directly in the discrete setting through optimal transport plans and their barycentric projections.

Usage

```
rbaryGD(
  atoms,
  marginals = NULL,
  weights = NULL,
  num_support = 100,
  alpha = 1,
  ...
)
```

Arguments

<code>atoms</code>	a length- K list where each element is an $(N_k \times P)$ matrix of atoms.
<code>marginals</code>	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.
<code>weights</code>	weights for each individual measure; if NULL (default), each measure is considered equally. Otherwise, it should be a length- K vector.
<code>num_support</code>	the number of support points M for the barycenter (default: 100).
<code>alpha</code>	step size parameter $\alpha \in (0, 1]$ controlling the update of support points (default: 1).
<code>...</code>	extra parameters including abstol stopping criterion for iterations (default: 1e-6). maxiter maximum number of iterations (default: 10). init.support optional matrix of initial support locations. candidate.mass optional fixed masses at the candidate locations. rtol, stationarity_tol relative change and relocation residual tolerances. inner_maxiter, max_backtrack, print.progress inner pivot budget, backtrack-ing budget, and progress flag.

Details

The default initializer selects systematic quantiles of pooled atom indices weighted by both input masses and outer weights. It does not require a full-rank sample covariance. Use ‘init.support’ to reproduce a chosen start. Returned histories contain accepted iterates including initialization. A small relocation residual is a local stopping criterion, not global optimality.

Value

A list including numerical status, objective, diagnostics, and:

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

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

history a vector of objective values over iterations.

alpha the step size used for the update.

niter the number of completed iterations.

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](https://doi.org/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 iteratively-reweighted least-squares (IRLS) majorization idea according to You et al. (2025). Each surrogate receives four exact-plan relocation steps, not a guarantee of a globally solved barycenter subproblem. Accepted outer steps are checked against the full objective; local stopping does not certify global optimality.

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 outer iterations (default: 10).

init.support initial numeric support matrix; otherwise a deterministic weighted systematic selection of pooled atoms is used.

candidate.mass fixed candidate masses; default uniform.

smoothing nonnegative median smoothing parameter (default: zero). A positive value changes the target to the sum of square roots of squared distances plus squared smoothing.

rtol relative support-change tolerance (default: 1e-6).

stationarity_tol relocation-residual tolerance (default: 1e-7).

alpha relocation step fraction in (0,1] (default: one).

inner_maxiter exact OT optimization-pivot budget (default: 100000000); zero is unlimited.

max_backtrack maximum objective-based step halvings (default: 12).

print.progress show objective progress (default: FALSE).

Value

A result list retaining the legacy fields below, and adding objective, status, stopping reason, initialization, controls, and numerical diagnostics. Non-success states, including an unresolved collision, issue a warning. The feasible set has moving support locations with fixed masses. Scalar point-mass medians use a weighted-median certificate; more general collisions require a metric dominance certificate or a verified descent escape, otherwise their unresolved status remains visible.

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

weight a length- M vector of median's fixed weights (uniform unless supplied).

history accepted objective values, including initialization and genuine zeros.

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 approximated using a single barycentric-projection relocation of an IRLS majorizer per outer step. The conditional mean of a coupling is a barycentric projection; it need not be a transport map. Accepted steps are checked against the full median objective.

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 outer iterations (default: 10). init.support initial numeric support matrix; otherwise a deterministic weighted systematic selection of pooled atoms is used. candidate.mass fixed candidate masses; default uniform. smoothing nonnegative median smoothing parameter (default: zero). A positive value changes the target to the sum of square roots of squared distances plus squared smoothing. rtol relative support-change tolerance (default: 1e-6). stationarity_tol relocation-residual tolerance (default: 1e-7). alpha relocation step fraction in $(0,1]$ (default: one). inner_maxiter exact OT optimization-pivot budget (default: 100000000); zero is unlimited. max_backtrack maximum objective-based step halvings (default: 12). print.progress show objective progress (default: FALSE).

Value

A result list retaining the legacy fields below, and adding objective, status, stopping reason, initialization, controls, and numerical diagnostics. Non-success states, including an unresolved collision, issue a warning. The feasible set has moving support locations with fixed masses. Scalar point-mass medians use a weighted-median certificate; more general collisions require a metric dominance certificate or a verified descent escape, otherwise their unresolved status remains visible.

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

weight a length- M vector of median's fixed weights (uniform unless supplied).

history accepted objective values, including initialization and genuine zeros.

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 ν . Then, the regularized problem adds a penalty term to the objective function:

$$F_{p,\lambda}(\mu, \nu) = \min_{\Gamma \in \Pi(\mu, \nu)} \langle \Gamma, C \rangle + \lambda \sum_{m,n} \Gamma_{m,n} (\log(\Gamma_{m,n}) - 1),$$

where $\lambda > 0$ is the regularization parameter and Γ denotes a transport plan, with zero entropy contribution at zero plan entries. For unit total mass, omitting the minus-one term increases the objective by λ without changing its minimizer. The returned `regularized_objective` follows the displayed convention; distance is the rooted transport component. As $\lambda \rightarrow 0$, the regularized solution converges to the exact Wasserstein solution, but small values of λ can require many iterations. The solver uses log-domain updates and reports iteration limits and numerical failures in the result, with a warning. Nonfinite full objectives are numerical failures even if the returned plan satisfies the marginal tolerance. The separate `marginal_converged` field records that marginal check. Invalid inputs cause errors. Inspect the status and residuals before using a computed value; do not interpret a finite-budget approximation as exact OT.

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 probability mass vector that sums to 1. If NULL (default), uniform weight is set.
wy	a length- N probability mass vector 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 The p th root of the transport component of the regularized plan; not an exact Wasserstein metric.

plan The approximate regularized transport plan. Status, marginal residuals, transport cost, and the full regularized objective are also returned.

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 distance compares univariate projections of probability measures. For the d -dimensional probability measures μ and ν , the SW distance is defined as

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

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 approximated by an equal-weight average of powered projected distances, followed by the p -th root. Each projected finite-measure problem is solved by exact monotone transport. A finite collection of directions can define only a pseudometric; no sphere-integration error estimate is returned.

Usage

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

Arguments

X an $(M \times P)$ matrix of row observations or a finite measure from [ot_measure()].

Y an $(N \times P)$ matrix of row observations or a finite measure from [ot_measure()].

p a finite Wasserstein order at least one (default: 2).

... extra parameters including

- num_proj** number of directions (default: 496, or one in 1D).
- directions** optional matrix with one nonzero direction per row and one column per coordinate; rows are normalized to unit length. A numeric vector specifies one direction. If num_proj is also supplied, it must match.
- wx,wy** optional nonnegative masses on the rows of X and Y, normalized separately; defaults are uniform. For measure objects, their stored masses are used and the corresponding mass argument must be NULL.

Value

a named list containing

distance p -th root of the mean powered projected distances.

projdist a vector of projected univariate distances.

directions unit directions actually used, one per row, permitting replay.

approximation exact_1d or finite_direction_average.

status,stop_reason,diagnostics finite-solver status and direction metadata; success does not certify accuracy of integration over the sphere.

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](https://doi.org/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 = seq_along(outsw$projdist)
plot_y = (base::cumsum(outsw$projdist^2)/plot_x)^(1/2)

# 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)
```

Description

This function computes the $\mathcal{W}_p$ distance between two empirical measures and returns descriptive resampled distances. It resamples support-row indices with the supplied probabilities, using the

number of rows as the resample size. This is not a generally calibrated confidence interval or hypothesis test. Coalescing atoms changes that sample-size convention; survey weights and hierarchical sampling require separate justification.

Usage

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

Arguments

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

```
#-----
# Bootstrapping 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 probability mass vector that sums to 1. If NULL (default), uniform weight is set.
wy	a length- N probability mass vector 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 $(\mu_1, \mu_2, \dots, \mu_M)$ and $(\nu_1, \nu_2, \dots, \nu_N)$ 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:10.1561/22000000073.

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

- * **datasets**
 - digit3, [3](#)
 - digits, [4](#)
- * **data**
 - digit3, [3](#)
 - digits, [4](#)
- * **dist**
 - ipot, [47](#)
 - pwdist, [60](#)
 - sinkhorn, [72](#)
 - swdist, [74](#)
 - wassboot, [75](#)
 - wasserstein, [77](#)
- * **ecdf**
 - ecdfbary, [5](#)
 - ecdfmed, [6](#)
- * **fixed_centroid**
 - fbary14C, [8](#)
 - fbary15B, [10](#)
- * **free_centroid**
 - pwbar, [57](#)
 - rbary23L, [63](#)
 - rbaryGD, [64](#)
 - rmedIRLS, [67](#)
 - rmedWB, [69](#)
- * **gaussian**
 - gaussbary1d, [14](#)
 - gaussbarypd, [15](#)
 - gaussmed1d, [17](#)
 - gaussmedpd, [18](#)
- * **gromov**
 - gwbary, [22](#)
 - gwdist, [23](#)
- * **histogram**
 - histbary, [26](#)
 - histbary14C, [27](#)
 - histbary15B, [29](#)
 - histdist, [31](#)
 - histinterp, [32](#)
- histmed, [34](#)
- * **image**
 - imagebary, [36](#)
 - imagebary14C, [38](#)
 - imagebary15B, [40](#)
 - imagedist, [42](#)
 - imageinterp, [43](#)
 - imagemed, [44](#)
- * **other**
 - fiedler, [12](#)
 - gaussvis2d, [20](#)
 - img2measure, [45](#)
- as_ot_measure (ot_measure), [55](#)
- digit3, [3](#)
- digits, [4](#)
- dist, [24](#)
- ecdfbary, [5](#)
- ecdfmed, [6](#)
- fbary14C, [8](#), [29](#), [39](#), [40](#)
- fbary14Cdist (fbary14C), [8](#)
- fbary15B, [10](#), [30](#), [41](#)
- fbary15Bdist (fbary15B), [10](#)
- fiedler, [12](#)
- gaussbary1d, [14](#)
- gaussbarypd, [15](#)
- gaussmed1d, [17](#)
- gaussmedpd, [18](#)
- gaussvis2d, [20](#)
- gwbary, [22](#)
- gwdist, [22](#), [23](#)
- histbary, [26](#)
- histbary14C, [27](#)
- histbary15B, [29](#)
- histdist, [31](#)
- histinterp, [32](#)

histmed, [34](#)

imagebary, [36](#)
imagebary14C, [38](#)
imagebary15B, [40](#)
imagedist, [42](#), [45](#)
imageinterp, [43](#)
imagemed, [44](#)
img2measure, [45](#)
ipot, [47](#)
ipotD (ipot), [47](#)

ot_collection (ot_measure), [55](#)
ot_cost, [49](#)
ot_density (ot_image), [53](#)
ot_distance, [49](#)
ot_gaussian, [50](#)
ot_gaussian_distance, [51](#)
ot_gaussian_summary, [52](#)
ot_image, [53](#)
ot_inspect, [54](#)
ot_measure, [55](#)
ot_summary, [56](#)

pwbary, [57](#)
pwdist, [60](#)

quantile.t4transport_cdf, [62](#)

rbary23L, [63](#)
rbaryGD, [64](#)
rmedIRLS, [67](#)
rmedWB, [69](#)

sinkhorn, [72](#)
sinkhornD (sinkhorn), [72](#)
swdist, [74](#)

wassboot, [75](#)
wasserstein, [77](#)
wassersteinD (wasserstein), [77](#)