Package ‘WSGeometry’

December 9, 2020

Type Package

Title Compute Wasserstein Barycenters, Geodesics, Principal Components and Distances

Version 1.0

Date 04.12.2020

Author Florian Heinemann

Maintainer Florian Heinemann <florian.heinemann@uni-goettingen.de>

Description Compute Wasserstein Barycenters, Geodesics, Principal Components and Distances.

License GPL-3

Encoding UTF-8

LazyData True

Imports Rcpp,RcppArmadillo, transport, RSpectra, expm, Rsymphony, plot3D, imager, grDevices, stats, graphics

Suggests magick, mvtnorm

LinkingTo RcppArmadillo, Rcpp

RoxygenNote 6.1.1

R topics documented:

barycenter_lp *Exact computation of 2-Wasserstein barycenters in R^d using linear programming*

Description

This function solves the 2-Wasserstein barycenter problem of N finitely supported input measures explicitly by solving the corresponding linear program.

Usage

```
barycenter_lp(pos.list, weights.list, frechet.weights = NULL)
```

Arguments

`pos.list` A list of Mxd matrices, specifying the positions of the data measures.

`weights.list` A list of vectors with non-negative entries and identical total sum specifying the weights of the data measures.

`frechet.weights` A vector of positive entries summing to one specifying the weights of each data measure in the Fréchet functional.

Value

A list with two entries. The first entry contains the positions of the computed barycenter and the second entry contains the corresponding weights.

References

E Anderes, S Borgwardt, and J Miller (2016). Discrete Wasserstein barycenters: Optimal transport for discrete data. Mathematical Methods of Operations Research, 84(2):389-409.

S Borgwardt and S Patterson (2020). Improved linear programs for discrete barycenters. Informs Journal on Optimization 2(1):14-33.

Examples

```
## Not run:
pos.list<-vector("list",4)
weights.list<-vector("list",4)
pos.list[[1]]<-matrix(c(0,0,1,1,1,0,0,1),nrow=4,ncol=2)/10
pos.list[[2]]<-matrix(c(9,9,10,10,10,9,9,10),nrow=4,ncol=2)/10
pos.list[[3]]<-matrix(c(9,9,10,10,1,0,0,1),nrow=4,ncol=2)/10
pos.list[[4]]<-matrix(c(0,0,1,1,10,9,9,10),nrow=4,ncol=2)/10
plot(0, 0, xlab = "", ylab = "", type = "n", xlim = c(0, 1), ylim = c(0, 1))
for(i in 1:4)
  points(pos.list[[i]][,1], pos.list[[i]][,2], col = i)
weights.list[[1]]<-rep(1/4,4)
weights.list[[2]]<-rep(1/4,4)
weights.list[[3]]<-rep(1/4,4)
weights.list[[4]]<-rep(1/4,4)
bary<-barycenter_lp(pos.list,weights.list)
points(bary$positions[,1],bary$positions[,2], col = "orange", pch = 13)
```

```
## End(Not run)
```

geodesic_pos *Compute Wasserstein geodesics*

Description

Computes the geodesic between two measures P1 and P2 in arbitrary dimensions at given time-points.

Usage

```
geodesic_pos(P1, P2, p = 2, steps)
```

Arguments

P1	One of the following: A matrix, representing an image; A file name containing an image; A wpp-object.
P2	One of the following: A matrix, representing an image; A file name containing an image; A wpp-object.
p	A real number >=1 specifying the exponent of the Wasserstein distance.
steps	A vector of numbers in [0,1] describing the time points at which the geodesic should be evaluated.

Value

A list of the same length as steps where each element is a wpp-object describing the value of the geodesic at the corresponding times in steps.

Examples

```
## Not run:
U<-runif(20)
U<-U/sum(U)
pos<-matrix(runif(2*20),nrow=20,ncol=2)
P1<-transport::wpp(pos,U)
U<-runif(20)
U<-U/sum(U)
pos<-matrix(runif(2*20),nrow=20,ncol=2)
P2<-transport::wpp(pos,U)
geodesic<-geodesic_pos(P1,P2,p=2,seq(0,1,0.1))
plotGeodesic(geodesic,File="GeodesicR2")

## End(Not run)
```

`location_scatter_bary` *Computes the 2-Wasserstein barycenter of location-scatter families of measures*

Description

This function solves the 2-Wasserstein barycenter problem of N measures from a location-scatter family, where each data distribution is given by a mean vector and a covariance matrix. In particular, this can be used to compute the barycenter of Gaussian distributions.

Usage

```
location_scatter_bary(means, cov, thresh = 10^(-5), maxiter = 100,
  showIter = FALSE)
```

Arguments

`means`	A list of mean vectors of the elements of the location-scatter family.
`cov`	A list of semipositive-definite covariance matrices of the elements of the location-scatter family.
`thresh`	A real number specifying the threshold for terminating the iterative algorithm.
`maxiter`	An integer specifying after how many iterations the algorithm should be terminated even if the specified threshold has not been reached.
`showIter`	A boolean specifying whether the number of performed iterations should be shown at the end.

Value

A list of two elements. The first element "mean" gives the mean vector of the barycenter measure. The second elemenent "cov" gives the covariance matrix of the barycenter measure.

References

PC Álvarez-Esteban, E del Barrio, JA Cuesta-Albertos, and C Matrán (2016). A fixed-point approach to barycenters in Wasserstein space. J. Math. Anal. Appl., 441(2):744–762.
Y Zemel and VM Panaretos (2019). Fréchet Means and Procrustes Analysis in Wasserstein Space. Bernoulli 25(2):932-976.

Examples

```
## Not run:
#One dimensional example
mean.list<-list(5,15)
var.list<-list(1,4)
res<-location_scatter_bary(mean.list,var.list)
x<-seq(0,22,10^-4)
y1<-dnorm(x,mean=5,sd=1)
y2<-dnorm(x,mean=15,sd=2)
y3<-dnorm(x,mean=res$mean,sd=sqrt(res$cov))
plot(x,y1,type="l",main = "Barycenter of two 1-d Gaussian distributions",
ylab = "density",col="blue")
```

```
lines(x,y2,col="green")
lines(x,y3,col="red")
legend(15,0.4, legend=c("N(5,1)", "N(15,4)","Barycenter"),
       col=c("blue","green","red"),lty=1,cex=0.9)


#two dimensional example
# packages graphics and mvtnorm are required to run this example
set.seed(2898581)
mean.list <- list(c(0,0), c(0,0), c(0,0))
COV <- 0.3 + rWishart(3, df = 2, Sigma = diag(2))
cov.list <- list(COV[,, 1], COV[,, 2], COV[,, 3])
res<-location_scatter_bary(mean.list, cov.list)

x <- y <- seq(-3, 3, .1)
z <- array(0.0, dim = c(length(x), length(y), 4))
for(i in seq_along(x))
 for(j in seq_along(y))
 {
   for(n in 1:3)
     z[i, j, n] <- mvtnorm::dmvnorm(c(x[i], y[j]), sigma = COV[, , n])

   z[i, j, 4] <- mvtnorm::dmvnorm(c(x[i], y[j]), sigma = res$cov)
 }

op <- par(mfrow = c(2, 2),  mai = c(0, 0, 0, 0))
for(n in 1:3)
{
 graphics::persp(x, y, z[, , n], theta = 30, phi = 30, expand = 0.5, col = "lightblue",
  zlab = "", ticktype = "detailed", shade = .75, lphi = 45, ltheta = 135)
 text(x = 0, y = 0.2, labels = paste("COV[,, ", n, "]", sep = ""))
}

graphics::persp(x, y, z[, , 4], theta = 30, phi = 30, expand = 0.5, col = "red", zlab = "",
ticktype = "detailed", shade = .75, lphi = 45, ltheta = 135)
text(x = 0, y = 0.2, labels = "barycenter")
par(op)

## End(Not run)
```

`multi_marginal` *Solve the multimarginal optimal transport problem by linear programming*

Description

Solves the N-fold multimarginal optimal transport problem between N specified measures and a specified cost. This is essentially a convenient wrapper function that builds and solves the corresponding linear program.

Usage

```
multi_marginal(weights, costA)
```

Arguments

`weights`	A list of vectors specifying the weights of the marginal distributions. These vectors do not need to be of the same size.
`costA`	An array where the entry (i1,i2,...,iN) specifies the value of the cost functional for the point i1 in the first measure, i2 in the second measure and so on.

Value

A list with two entries. The first entry contains the optimal multicoupling in array form, and the second entry contains the cost of the optimal solution.

Examples

```
## Not run:
W<-list(rep(1,10),rep(1,10),rep(1,10))
C<-array(runif(10^3),c(10,10,10))
MM<-multi_marginal(W,C)

## End(Not run)
```

`plotGeodesic` *Plot previously computed Wasserstein geodesics*

Description

This function generates either a sequence of images or a gif displaying the input optimal transport geodesic.

Usage

```
plotGeodesic(Geodesic, method = "default", images = TRUE, gif = TRUE,
  File = "Geodesic", resolution = c(400, 400), dotsize = 2,
  gridsize = c(100, 100), out.col = grey(0:1000/1000),
  splitrange = c(1, 1), turn = FALSE, phi = 40, theta = 40,
  fps = NULL)
```

Arguments

`Geodesic`	A list of measures in R^2 or R^3 corresponding to the discretized time steps of the geodesic. Each entry in the list must be one of the following: a pp-object; a wpp-object; or a list containing an entry named 'positions' that is an Mxd matrix.
`method`	A string specifying the method used to plot each measure. The input "default" generates a scatterplot where the size of each point corresponds to its mass. The input "bin" maps the data to a grid of prespecified size to plot it as an image. The input "binSplit" also maps the data to a grid, but afterwards the mass of each pixel is split between all pixels in a specified range. This generates smoother output images.
`images`	A boolean specifying whether image files should be generated.

`gif`	A boolean specifying whether a gif should be generated. To use this option the ImageMagick software and the magick R package need to be installed. (see https://cran.r-project.org/web/packages/magick/vignettes/intro.html for details and instructions.)
`File`	A string specifying the prefix of all generated files.
`resolution`	A vector with two elements specifying the resolution of the output images.
`dotsize`	A positive number working as a multiplier on the size of the dots in the output generated by the default method.
`gridsize`	A vector with two elements specifying the size of the grid used for "bin" and "binSplit".
`out.col`	A colour vector, specifying the colour scheme used in the call of the image function when plotting the output of the "bin" and "binSplit" methods. See the documentation of the image function for more details.
`splitrange`	A vector of two positive integers specifying the number of pixels the mass of a point is shared with in the "binSplit" method. The first entry controls the horizontal direction and the second controls the vertical direction.
`turn`	A boolean specifying whether the image should be rotated to account for the output of the image function.
`phi`	A number specifying the viewing direction in the three dimensional method. It gives the colatitude of the plot.
`theta`	A number specifying the viewing direction in the three dimensional setting. It gives the azimuthal direction of the plot.
`fps`	A positive number specifying the number of frames per second in the output gif. The default adjusts the fps to output a gif file of length 10 seconds.

Value

This function does not provide any return value, but instead generates output files in the current working directory.

Examples

```
## Not run:
#2D-Example:
library(transport)
set.seed(420)
N<-2
supp.size<-10^2
L<-sqrt(supp.size)
d<-2
data.list<-vector("list",N)
image.list<-vector("list",N)
for (i in 1:N){
  t.vec<-seq(0,2*pi,length.out=supp.size)
  pos<-cbind(cos(t.vec)*runif(1,0.2,1),sin(t.vec)*runif(1,0.2,1))
  theta<-runif(1,0,2*pi)
  rotation<-matrix(c(cos(theta),sin(theta),-1*sin(theta),cos(theta)),2,2)
  pos<-pos%*%rotation
  pos<-pos+1
  pos<-pos/2
  W<-rep(1/supp.size,supp.size)
  data.list[[i]]<-transport::wpp(pos,W)
```

```
}
Geo<-geodesic_pos(data.list[[1]],data.list[[2]],2,seq(0,1,0.1))
plotGeodesic(Geo,File="TestGeodesicDefault")
plotGeodesic(Geo,method="bin",File="TestGeodesicDefaultBin")
plotGeodesic(Geo,method="binSplit",File="TestGeodesicDefaultBinSPlit")

#3D-Example:
set.seed(123)
M<-40
U<-runif(1,0.5,1)
Torus<-WSGeometry:::gen_torus(M,U,min(U/2,runif(1)))
v<-WSGeometry:::normalize(runif(3))
Torus<-WSGeometry:::rotate3D(Torus,v,runif(1,0,2*pi))
Torus1<-Torus%*%diag(runif(3,1,3))
U<-runif(1,0.5,1)
Torus<-WSGeometry:::gen_torus(M,U,min(U/2,runif(1)))
v<-WSGeometry:::normalize(runif(3))
Torus<-WSGeometry:::rotate3D(Torus,v,runif(1,0,2*pi))
Torus2<-Torus%*%diag(runif(3,1,3))
L<-length(Torus)/3
Torus1<-transport::wpp(Torus1,rep(1/L,L))
Torus2<-transport::wpp(Torus2,rep(1/L,L))
geo<-geodesic_pos(Torus1,Torus2,p=2,seq(0,1,0.1))
plotGeodesic(geo,File="3dGeodesic")

## End(Not run)
```

`wasserstein_bary` *Compute Wasserstein barycenters*

Description

This function computes the Wasserstein barycenter of a list of suitable objects and returns the barycenter in a prespecified form.

Usage

```
wasserstein_bary(data.list, frechet.weights = NULL,
  method = "alternating", return_type = "wpp", supp.size = NULL,
  output.supp = NULL, shared = FALSE, sample.size = NULL,
  maxIter = 10, weights_maxIter = 100, pos_maxIter = 100,
  stepsize = 0.1, thresh = 10^(-12), regular = 10^-3,
  warmstart = TRUE, warmstartlength = 2, showIter = FALSE,
  threads = 1)
```

Arguments

`data.list` A list of objects of which the barycenter should be computed. Each element should be one of the following: A matrix, representing an image; A path to a file containing an image; A wpp-object; A pp-object; A list containing an entry named ‘positions‘ with the support of the measure and an entry named ‘weights‘ containing the weights of the support points; A list containing en entry named ‘positions“ specifying the support of a measure with uniform weights.

frechet.weights
: A real vector summing to 1, specifying the weights in the Frechet functional. Should be of the same length as data.list.

method
: A string specifiying the method to be used. This also determines which of the other parameters are active/used in this function call. See details for the specific methods currently available.

return_type
: A string specifying the format of the output. The currently available options are "default" (which gives list with entries ‘positions‘ and ‘weights‘); "wpp"- which gives a wpp-object; and "image_mat" for a matrix of the same dimensions as the input matrices (only for the regular method).

supp.size
: A positive integer specifying the size of the support used to approximate the barycenter in the "alternating" method.

output.supp
: An Mxd matrix specifying the support set on which the optimal weights of the barycenter should be approximated when method = "fixed_support". Each row of the matrix represents one support point in R^d.

shared
: A boolean flag specifying whether all measures have the same support set and the weights of the barycenter should be optimised over this set as well.

sample.size
: A positive integer specifying the number of samples drawn in the stochastic approximation of the barycenter for method "sampling".

maxIter
: A positive integer specifyng the maximum number of "outer" iterations. The full number of iteration steps performed is maxIter * (weights_maxIter+pos_maxIter).

weights_maxIter
: A positive integer specifying the maximum number of iterations on the weights of the barycenter.

pos_maxIter
: A positive integer specifying the maximum number of iterations on the support of the barycenter.

stepsize
: A positive number specifying the stepsize in the position iterations.

thresh
: A positive number specifying the minimal amount of change between iterations, which does not cause the algorithm to terminate.

regular
: A positive number specifying the regularisation parameter in the "regular" method.

warmstart
: A boolean specifying whether the algorithm should use a warmstart based on a stochastic subgradient descent.

warmstartlength
: A positive integer specifying the length of the warmstart. The number of steps in the SGD in the warmstart is ‘length(data.list)*warmstartlength‘.

showIter
: A boolean specifying whether the number of "outer" iterations performed should be shown at the end.

threads
: A positive integer specifying the number of threads used for parallel computing.

Details

This is the main function of this package. It computes/approximates 2-Wasserstein barycenters using the different methods outlined in the following.
"lp". Here the barycenter problem can be posed as a linear program. This method builds and solves this linear program. While this gives exact solutions to the problem, this method is highly run-time extensive and should only be used on small datasets. (See Anderes et al. (2016) and Borgwardt & Patterson (2020) for details).
"regular". This method solves the entropy-regularised fixed support barycenter problem. Here, a

penalisation term is introduced to the problem, which yields a strictly convex problem that can be solved with Sinkhorn's algorithm (for details see Benamou et al. (2015)). Additionally, it is assumed that all the measures have the same support set, and instead of an exact (regularised) barycenter, the methods finds the best solution having the same support as the data. This is quite reasonable when the dataset consists of images and the barycenter should be an image as well. The choice of the regularisation parameter "regular" is a delicate issue. Large values reduce the runtime, but yield "blurry" barycenters. Small values yield sharper results, but have longer run-time and may cause numerical instabilities. The choice of this parameter depends on the dataset at hand, and will typically require tuning.
"fixed_support". This method computes the best approximation of the barycenter, which is supported on a pre-specified support set (as supplied by the parameter "output.support"). Contrary to the "regular" method, here this set does not need to coincide with any of the support sets of the data measures. See Cuturi & Doucet (2014) for details.
"alternating". This method computes the best approximation of the barycenter with a certain support size. It alternates between finding the best positions for given weights and then finding the best weights for these positions. See Cuturi and Doucet (2014) for details.
"sampling". This method uses the SUA method of Heinemann et al. (2020) to generate a stochastic approximation of the barycenter. It replaces the original measures by empirical measures obtained from samples of size 'sample.size' from each data measure.

References

E Anderes, S Borgwardt, and J Miller (2016). Discrete Wasserstein barycenters: Optimal transport for discrete data. Mathematical Methods of Operations Research, 84(2):389-409.
S Borgwardt and S Patterson (2020). Improved linear programs for discrete barycenters. Informs Journal on Optimization 2(1):14-33.
J-D Benamou, G Carlier, M Cuturi, L Nenna, and G Peyré (2015). Iterative Bregman projections for regularized transportation problems. SIAM Journal on Scientific Computing 37(2):A1111-A1138.
M Cuturi and A Doucet (2014). Fast Computation of Wasserstein Barycenters. Proceedings of the 31st International Conference on Machine Learning, PMLR 32(2):685-693.
F Heinemann, A Munk, and Y Zemel (2020). Randomised Wasserstein barycenter computation: Resampling with statistical guarantees. arXiv preprint.

Examples

```
## Not run:
##Basic Examples
#build list
K<-1
N<-4*K
M<-16
d<-2
data.list<-vector("list",N)

###image_mat
for (i in 1:K){
  U<-runif(M)
  U<-U/sum(U)
  data.list[[i]]<-matrix(U,sqrt(M))
}

#wpp
for (i in (K+1):(2*K)){
  U<-runif(M)
  U<-U/sum(U)
```

```
  pos<-matrix(runif(d*M),M,d)
  data.list[[i]]<-transport::wpp(pos,U)
}

#point pattern
for (i in (2*K+1):(3*K)){
  pos<-matrix(runif(d*M),M,d)
  data.list[[i]]<-list(positions=pos)
}

#weighted point pattern

for (i in (3*K+1):(4*K)){
  U<-runif(M)
  U<-U/sum(U)
  pos<-matrix(runif(d*M),M,d)
  data.list[[i]]<-list(positions=pos,weights=U)
}

system.time(res1<-wasserstein_bary(data.list,return_type = "wpp",method="lp"))
WSGeometry:::frechet_func(res1,data.list)
system.time(res2<-wasserstein_bary(data.list,return_type = "wpp",method="alternating",
supp.size = M*N-N+1,warmstartlength = 3,pos_maxIter = 100,weights_maxIter = 100))
WSGeometry:::frechet_func(res2,data.list)
system.time(res3<-wasserstein_bary(data.list,return_type = "wpp",
method="fixed_support",warmstartlength = 3,weights_maxIter = 100,output.supp = res1$coordinates))
WSGeometry:::frechet_func(res3,data.list)
system.time(res4<-wasserstein_bary(data.list,return_type = "wpp",
method="sampling",sample.size=8,warmstartlength = 3,pos_maxIter = 100))
WSGeometry:::frechet_func(res4,data.list)
##Visual Examples
###ellipses
set.seed(420)
N<-20
supp.size<-10^2
L<-sqrt(supp.size)
d<-2
data.list<-vector("list",N)
image.list<-vector("list",N)
for (i in 1:N){
  t.vec<-seq(0,2*pi,length.out=supp.size)
  pos<-cbind(cos(t.vec)*runif(1,0.2,1),sin(t.vec)*runif(1,0.2,1))
  theta<-runif(1,0,2*pi)
  rotation<-matrix(c(cos(theta),sin(theta),-1*sin(theta),cos(theta)),2,2)
  pos<-pos%*%rotation
  pos<-pos+1
  pos<-pos/2
  W<-rep(1/supp.size,supp.size)
  data.list[[i]]<-transport::wpp(pos,W)
  I<-WSGeometry:::bin2d(data.list[[i]]$coordinates,data.list[[i]]$mass,c(L*2,L*2))
  I<-WSGeometry:::smear(I,1,1)
  I<-I/sum(I)
  image.list[[i]]<-I
}

system.time(res1<-wasserstein_bary(data.list,return_type = "wpp",method="alternating"
,supp.size = supp.size,warmstartlength = 0,pos_maxIter = 10,weights_maxIter = 10,maxIter = 10))
```

```
plot(res1)
system.time(res2<-wasserstein_bary(data.list,return_type = "wpp",
method="fixed_support",warmstartlength = 0,weights_maxIter = 50,maxIter=1,
output.supp = WSGeometry:::grid_positions(2*L,2*L)))
plot(res2)
system.time(res3<-wasserstein_bary(data.list,return_type = "wpp",method="sampling",
sample.size=400,warmstartlength = 0,pos_maxIter = 100,stepsize = 1,maxIter=1))
plot(res3)

system.time(res4<-wasserstein_bary(image.list,return_type = "wpp",
method="regular",stepsize = 1,weights_maxIter = 50))
plot(res4)
system.time(res5<-wasserstein_bary(image.list,return_type = "wpp",
method="fixed_support",shared=TRUE,warmstartlength = 0,weights_maxIter = 50,maxIter=1,
output.supp = WSGeometry:::grid_positions(2*L,2*L)))
plot(res5)

## End(Not run)
```

`ws_dist` *Compute the p-Wasserstein distance between two measures*

Description

This is essentially a wrapper function of transport. It has the advantage of allowing more general input objects, such as images or matrices, without the user having to manually convert these objects.

Usage

```
ws_dist(A, B, p = 2, sampling = FALSE, S = NULL, R = NULL)
```

Arguments

A	One of the following: A matrix, representing an image; A file name containing an image; A wpp-object.
B	One of the following: A matrix, representing an image; A file name containing an image; A wpp-object.
p	A positive real number specifying the power of the Wasserstein distance.
sampling	A boolean specifying whether a stochastic approximation (Sommerfeld et al., 2019) should be used to approximate the distance.
S	A positive integer specifying the number of samples drawn in the stochastic approximation.
R	The number of repetitions averaged over in the stochastic approximation.

Value

A number specifying the computed p-Wasserstein distance.

References

M Sommerfeld, J Schrieber, Y Zemel, and A Munk (2019). Optimal transport: Fast probabilistic approximations with exact solvers. Journal of Machine Learning Research 20(105):1–23.

Examples

```
## Not run:
P1<-transport::random32a$mass
P2<-transport::random32b$mass
P1<-P1/sum(P1)
P2<-P2/sum(P2)
ws_dist(P1,P2)

## End(Not run)
```

---

ws_logpca — *Computes Wasserstein principal components*

Description

Computes principal components in the 2-Wasserstein Space for a dataset of weighted point measures in R^2.

Usage

```
ws_logpca(data.list, barycenter, pca.count, steps_number = 21)
```

Arguments

`data.list`	A list of objects of which the principal components should be computed. Each element should be one of the following: A matrix, representing an image; A file name containing an image; A wpp-object; A pp-object; A list containing an entry 'positions' with the support of the measure and an entry 'weights' containing the weights of the support points; A list containing en entry 'positions' specifying the support of a measure with uniform weights.
`barycenter`	A barycenter of the dataset. See data.list for possible object types.
`pca.count`	An integer specifying the number of principal components to be computed.
`steps_number`	An integer specifying the number of discretisation steps for the output of the principal components.

Details

This function computes the principal components of a dataset consisting of weighted point measures in R^2. To do this it first maps the data to the tangent space at the barycenter and then performs standard Euclidean PCA on this space. Afterwards the resulting components are mapped back to the 2-Wasserstein space.

Value

A list with two entries. The first contains a list (of length pca.count), where each entry is a list of wpp-objects specifying one point on the corresponding principal component. The second entry is a vector containing the eigenvalues of the computed principal components. The output can be plotted by applying plotGeodesic to each component of the list.

References

E Cazelles, V Seguy, J Bigot, M Cuturi, and N Papadakis (2017); Log-PCA versus Geodesic PCA of histograms in the Wasserstein space. SIAM Journal on Scientific Computing 40(2):B429–B456.
W Wei, D Slepcev, S Basu, JA Ozolek, and GK Rohde (2013). A Linear Optimal Transportation Framework for Quantifying and Visualizing Variations in Sets of Images. International Journal of Computer Vision 101(2):254-269.

Examples

```
## Not run:
set.seed(2020)
N<-20
supp.size<-10^2
L<-sqrt(supp.size)
d<-2
data.list<-vector("list",N)
image.list<-vector("list",N)
for (i in 1:N){
  t.vec<-seq(0,2*pi,length.out=supp.size)
  pos<-cbind(cos(t.vec)*runif(1,0.2,1),sin(t.vec)*runif(1,0.2,1))
  theta<-runif(1,0,2*pi)
  rotation<-matrix(c(cos(theta),sin(theta),-1*sin(theta),cos(theta)),2,2)
  pos<-pos%*%rotation
  pos<-pos+1
  pos<-pos/2
  W<-rep(1/supp.size,supp.size)
  data.list[[i]]<-transport::wpp(pos,W)
  I<-WSGeometry:::bin2d(data.list[[i]]$coordinates,data.list[[i]]$mass,c(L*2,L*2))
  I<-WSGeometry:::smear(I,1,1)
  I<-I/sum(I)
  image.list[[i]]<-I
}

res1<-wasserstein_bary(data.list,return_type = "wpp",method="alternating",
supp.size = supp.size,warmstartlength = 0,pos_maxIter = 10,weights_maxIter = 10,maxIter = 10)
pcomps<-ws_logpca(data.list,res1,3)
plotGeodesic(pcomps$components[[1]],File="PCA1")
plotGeodesic(pcomps$components[[2]],File="PCA2")
plotGeodesic(pcomps$components[[3]],File="PCA3")

## End(Not run)
```

Index