Package: transport (via r-universe)

October 28, 2026

Version 0.15-4

Date 2024-09-15

Title Computation of Optimal Transport Plans and Wasserstein Distances

Maintainer Dominic Schuhmacher

<dominic.schuhmacher@mathematik.uni-goettingen.de>

Depends R (>= 4.1)

Imports grDevices, graphics, methods, stats, Rcpp (>= 0.12.10), data.table

Suggests animation, ks, testthat

LinkingTo RcppEigen, Rcpp

Description Solve optimal transport problems. Compute Wasserstein distances (a.k.a. Kantorovitch, Fortet--Mourier, Mallows, Earth Mover's, or minimal L_p distances), return the corresponding transference plans, and display them graphically. Objects that can be compared include grey-scale images, (weighted) point patterns, and mass vectors.

LazyData yes

Encoding UTF-8

License GPL (>= 2)

BugReports https://github.com/dschuhmacher/transport/issues

URL https://dschuhm1.pages.gwdg.de/software

RoxygenNote 7.3.1

Config/testthat/edition 3

NeedsCompilation yes

Author Dominic Schuhmacher [aut, cre], Björn Bähre [aut] (aha and power diagrams), Nicolas Bonneel [aut] (networkflow), Carsten Gottschlich [aut] (simplex and shortlist), Valentin Hartmann [aut] (semidiscrete1), Florian Heinemann [aut] (transport_track and networkflow integration), Bernhard Schmitzer [aut] (shielding), Jörn Schrieber [aut] (subsampling), Timo Wilm [ctb] (wpp)

Repository https://cran.r-universe.dev

Date/Publication 2024-09-17 02:35:01 UTC

RemoteUrl https://github.com/cran/transport

RemoteRef HEAD

RemoteSha 86b5d4ddb993f65a978fbaf8a79723dcc9e7fae1

Contents

transport-package . . . 3
aha . . . 4
all.equal (transport objects) . . . 6
compatible . . . 6
matimage . . . 7
methods . . . 8
pgrid . . . 9
pgrid-object . . . 10
plot . . . 11
plot.ut_pgrid . . . 12
plot.ut_wpp . . . 13
plot_apollonius . . . 14
power_diagram . . . 15
pp . . . 17
pp-object . . . 18
random . . . 18
ret_message . . . 19
semidiscrete . . . 20
semidiscrete1 . . . 21
shielding . . . 23
starting solutions . . . 25
subwasserstein . . . 26
transport . . . 28
transport_track . . . 34
trcontrol . . . 36
unbalanced . . . 37
wasserstein . . . 40
wasserstein1d . . . 41
wpp . . . 43
wpp-object . . . 44

Index **45**

transport-package *Optimal Transport in Various Forms*

Description

Solve optimal transport problems. Compute Wasserstein distances (a.k.a. Kantorovitch, Fortet–Mourier, Mallows, Earth Mover's, or minimal L_p distances), return the corresponding transport plans, and display them graphically. Objects that can be compared include grey-scale images, (weighted) point patterns, and mass vectors.

Details

Package:	transport
Type:	Package
Version:	0.12-1
Date:	2019-08-07
License:	GPL (>=2)
LazyData:	yes

The main end-user function is `transport`. It computes optimal transport plans between images (class `pgrid`), point patterns (class `pp`), weighted point patterns (class `wpp`) and mass vectors, based on various algorithms. These transport plans can be `plot`ed. The function `wasserstein` allows for the numerical computation of p-th order Wasserstein distances.

Most functions in this package are designed for data in two and higher dimensions. A quick tool for computing the p-th order Wasserstein distance between univariate samples is `wasserstein1d`.

Author(s)

Dominic Schuhmacher <schuhmacher@math.uni-goettingen.de>

Björn Bähre <bjobae@gmail.com> (code for `aha`-method)

Nicolas Bonneel <nicolas.bonneel@liris.cnrs.fr>
(adaptation of LEMON code for fast `networkflow` method)

Carsten Gottschlich <gottschlich@math.uni-goettingen.de>
(original java code for `shortlist` and `revsimplex` methods)

Valentin Hartmann <valentin.hartmann@epfl.ch> (code for `aha` method for p=1)

Florian Heinemann <florian.heinemann@uni-goettingen.de>
(integration of `networkflow` method)

Bernhard Schmitzer <schmitzer@uni-muenster.de> (`shielding` method)

Jörn Schrieber <joern.schrieber-1@mathematik.uni-goettingen.de> (`subsampling` method)

Maintainer: Dominic Schuhmacher <dominic.schuhmacher@mathematik.uni-goettingen.de>

References

See help page for the function transport.

Examples

```
##  See examples for function transport
```

---

aha *Solve Transportation Problem by Aurenhammer–Hoffmann–Aronov Method*

Description

Solve transportation problem by Aurenhammer–Hoffmann–Aronov Method.

Usage

```
aha(a, b, nscales = 1, scmult = 2, factr = 1e+05, maxit = 10000, powerdiag=FALSE,
    wasser = FALSE, wasser.spt = NA, approx=FALSE, ...)
transport_apply(a, tplan)
transport_error(a, b, tplan)
```

Arguments

a	an $m \times n$ matrix. a is treated as a measure on $[0, m] \times [0, n]$ with constant density on each unit square $[i, i+1) \times [j, j+1)$.
b	either a matrix such that dim(a)==dim(b) and sum(a)==sum(b) or a data frame with three variables named x, y and mass such that sum(a)==sum(b$mass), representing a discrete measure on $[0, m]) \times [0, n]$.
tplan	a transference plan from a (to b), typically an optimal transference plan obtained by a call to aha.
nscales, scmult	the number of scales to use for the multiscale approach (the default is 1 meaning no multiscale approach), and the factor by which the number of pixels in each dimension is multiplied to get from a coarser to the next finer scale.
factr, maxit	parameters passed to the underlying L-BFGS-B algorithm (via the argument control in the R-function optim).
powerdiag	logical. Instead of an optimal transference plan, should the parameters for the optimal power diagram be returned?
wasser	logical. Instead of an optimal transference plan, should only the L_2-Wasserstein-distance between a and b be returned?
wasser.spt	the number of support points used to approximate the discrete measure b. Defaults to NA meaning the full set of support points of b is used. If this argument is not NA, wasser is set to TRUE.
approx	logical. If TRUE, an approximation to the objective function is used during optimization.
...	further arguments passed to optim via its argument control.

Details

The function aha implements the algorithm by Aurenhammer, Hoffmann and Aronov (1998) for finding optimal transference plans in terms of the squared Euclidean distance in two dimensions. It follows the more detailed description given in Mérigot (2011) and also implements the multiscale version presented in the latter paper.

The functions `transport_apply` and `transport_error` serve for checking the accuracy of the transference plan obtained by aha. Since this transference plan is obtained by continuous optimization it will not transport exactly to the measure b, but to the measure `transport_apply(a, tplan)`. By `transport_error(a, b, tplan)` the sum of absolut errors between the transported a-measure and the b-measure is obtained.

Value

If `powerdiag` and `wasser` are both `FALSE`, a data frame with columns `from`, `to` and `mass`, which specify from which knot to which other knot what amount of mass is sent in the optimal transference plan. Knots are given as indices in terms of the usual column major enumeration of the matrices a and b. There are `plot` methods for the classes `pgrid` and `pp`, which can plot this solution.

If `powerdiag` is TRUE and `wasser` is `FALSE`, a list with components `xi`, `eta`, `w` and `rect`, which specify the parameters for the optimal power diagram in the same format as needed for the function `power_diagram`. Note that rect is always `c(0,m,0,n)`. Since version 0.10-0 the list has a further component `wasser.dist` containing the Wasserstein distance.

If `wasser` is `TRUE`, a data frame with columns `wasser.dist` and `error.bound` of length one, where `error.bound` gives a bound on the absolute error in the Wasserstein distance due to approximating the measure b by a measure on a smaller number of support points.

Author(s)

Björn Bähre <bjobae@gmail.com>
(slightly modified by Dominic Schuhmacher <dschuhm1@uni-goettingen.de>)

References

F. Aurenhammer, F. Hoffmann and B. Aronov (1998). Minkowski-type theorems and least-squares clustering. Algorithmica 20(1), 61–76.

Q. Mérigot (2011). A multiscale approach to optimal transport. Eurographics Symposium on Geometry Processing 30(5), 1583–1592.

See Also

`transport`, which is a convenient wrapper function for various optimal transportation algorithms.

Examples

```
# There is one particular testing configuration on MacOS where the following
# command does not return (to be investigated)
# res <- aha(random32a$mass, random32b$mass)
# plot(random32a, random32b, res, lwd=0.75)

aha(random64a$mass, random64b$mass, nscales=3, scmult=5, wasser.spt=512, approx=TRUE)
```

all.equal (transport objects)

Methods for Judging Near Equality of Objects of Class pgrid, pp or wpp

Description

Methods for judging near equality of objects of class pgrid or pp or wpp

Usage

```
## S3 method for class 'pgrid'
all.equal(target, current, ...)
## S3 method for class 'pp'
all.equal(target, current, ...)
## S3 method for class 'wpp'
all.equal(target, current, ...)
```

Arguments

target, current	the objects of the same class to be compared.
...	currently without effect.

Value

Either TRUE or a vector of mode "character" describing the differences between target and current.

Author(s)

Dominic Schuhmacher <dschuhm1@uni-goettingen.de>

See Also

all.equal (base), compatible

compatible

Test whether Two Objects are Compatible

Description

Test whether two objects of the same class are 'of similar shape' so that the function transport can be applied.

Usage

```
compatible(target, current, ...)
## S3 method for class 'pgrid'
compatible(target, current, ...)
## S3 method for class 'pp'
compatible(target, current, ...)
## S3 method for class 'wpp'
compatible(target, current, ...)
```

Arguments

target, current	to objects of the same class to be compared.
...	currently without effect.

Value

Logical.

Author(s)

Dominic Schuhmacher <dschuhm1@uni-goettingen.de>

See Also

all.equal

---

matimage *Plotting Matrices as Images*

Description

A simple wrapper to the image function with a more convenient syntax for plotting matrices "the right way round" as pixel images.

Usage

```
matimage(z, x = 1:dim(z)[1], y = 1:dim(z)[2], rot = TRUE, asp = 1, ...)
```

Arguments

z	a numeric matrix.
x, y	(optional) coordinates of the pixels.
rot	logical. Whether to plot the matrix "the right way round" so that the pixel position in the image corresponds to the pixel position in the matrix obtained by print.
asp	the aspect ratio parameter of image.
...	further parameters passed to image.

Value

Nothing (invisible NULL).

Examples

```
m <- matrix(1:36,6,6)
image(z=m, col = heat.colors(36))
matimage(m, col = heat.colors(36))
```

`methods` *Print and Summary Methods for Objects of Class pgrid, pp and wpp*

Description

Prints a brief description of a pixel grid or a point pattern.

Usage

```
## S3 method for class 'pgrid'
print(x, ...)
## S3 method for class 'pp'
print(x, ...)
## S3 method for class 'wpp'
print(x, ...)
## S3 method for class 'pgrid'
summary(object, ...)
## S3 method for class 'pp'
summary(object, ...)
## S3 method for class 'wpp'
summary(object, ...)
```

Arguments

`x`, `object`	an object of class pgrid or pp or wpp.
`...`	additional arguments. Currently without effect.

Details

Currently there is no difference between print and summary.

Author(s)

Dominic Schuhmacher <dschuhm1@uni-goettingen.de>
Timo Wilm <timo.wilm@stud.uni-goettingen.de>

pgrid *Constructor for the pgrid Class*

Description

Construct an object of class "pgrid" from a matrix or a higher-dimensional array.

Usage

```
pgrid(mass, boundary, gridtriple, generator, structure)
```

Arguments

mass	a matrix or higher-dimensional array specifing the masses in each pixel / at each pixel centre.
boundary, gridtriple, generator	arguments specifying the positions of the pixels. At most one of these can be specified.
structure	optional character string specifying the structure of the grid. Currently only "square" and "rectangular" make sense, and are derrived automatically from the dimensions of mass.

Details

For more detailed explanations of the arguments and other components of the derived object of class "pgrid", see pgrid-object.

Author(s)

Dominic Schuhmacher <dschuhm1@uni-goettingen.de>

See Also

Description of pgrid objects.

Examples

```
m <- matrix(1:20, 4, 5)
a <- pgrid(m)
print(a)
print.default(a)

## Not run:
  plot(a, rot=TRUE)
## End(Not run)
```

pgrid-object *Class of Pixel Grids*

Description

The class "pgrid" (for pixel grid) represents regular quantizations of measures on (bounded subsets of) R^d. Currently only square quantizations of measures on a rectangles are supported, which in 2-d can be thought of as grey scale images.

Details

Objects of class "pgrid" can be created by the function pgrid, and are most commonly used as input to the function transport. There are methods plot, print and summary for this class.

An object of class "pgrid" contains the following elements:

structure	the structure of the grid. Currently only "square" and "rectangular" are supported.
dimension	the dimension d of the space in which the grid is embedded. Must be ≥ 2.
n	the number of pixels along the various coordinates, a vector of length dimension.
N	the total number of pixels.
boundary	the outer boundary of the "picture" (i.e. of the support of the measure). A vector of length 2*dimension, where the odd entries contain the left and the even entries contain the right endpoints of the various coordinates.
gridtriple	the rule for generating the pixel centres along the various coordinates. A dim by 3 matrix where each row is of the form c(start, end, step).
generators	the pixel centres along the various coordinates. A list of length dim where the i-th element is a vector of length n[i].
mass	the array of masses in each pixel / at each pixel centre. In 2-d orientation corresponds to the standard orientation of images, see e.g. image. This means that pixels are arranged on coordinate axes in the order of their indices.

Author(s)

Dominic Schuhmacher <dschuhm1@uni-goettingen.de>

See Also

Constructor function pgrid.

plot *Methods for Plotting Objects of Class pgrid, pp and wpp*

Description

Methods for plotting objects of class pgrid, pp and wpp, possibly together with a transference plan.

Usage

```
  ## S3 method for class 'pgrid'
plot(x, y = NULL, tplan = NULL, mass = c("colour", "thickness"),
    length = 0.1, angle = 5, acol, bcol = 4, pcol="goldenrod2", lwd, pmass=TRUE,
    rot = FALSE, overlay = FALSE, static.mass =TRUE, ...)
  ## S3 method for class 'pp'
plot(x, y = NULL, tplan = NULL, cols = c(4, 2), cex = 0.8,
    acol = grey(0.3), lwd = 1, overlay = TRUE, ...)
  ## S3 method for class 'wpp'
plot(x, y = NULL, tplan = NULL, pmass=TRUE, tmass=TRUE, cols = c(4, 2),
    cex = 0.8, aglevel = 0.4, acol = grey(0.3), lwd = 1, overlay = TRUE, ...)
```

Arguments

x, y	one or two objects of class pgrid or class pp to be plotted.
tplan	a transference plan between the two objects x and y, typically an optimal transference plan obtained by a call to transport.
mass, pmass, tmass	for pgrid objects with a tplan: if mass == "colour", the mass transferred is depicted by heatmap colours; if mass == "thickness", it is depicted by the line widths of the arrows. For wpp objects: pmass, tmass are logicals controlling whether the *amount* of mass associated with the points and the mass transferred should be depicted in the plot.
length	the length of the arrow heads in inches.
aglevel	for wpp objects with tmass = TRUE: the grey level chosen for depicting the transport of an average amount of mass.
acol	the colour of the arrows/lines of the transference plan. Ignored for pgrid objects if mass = "colour" and for wpp objects if tmass is TRUE.
angle	the angle of the arrow heads.
bcol	the colour of the cell boundaries for a semidiscrete transport plan. Ignored in all other instances.
pcol	the colour of the points representing the discrete masses for a semidiscrete transport plan. Ignored in all other instances.
cols	for pp objects: A vector of size 2 specifying the colours of the two pp objects.

cex, lwd, ...	further graphic parameters used by plot. Note that for pgrid objects acol is ignored for mass == "colour", and lwd is ignored for mass == "thickness". Setting any of these parameters is optional.
rot	logical. Whether the mass matrices of pgrid objects should be rotated before calling image so that the orientation of the ploted pixelgrid and the orientation of the mass matrix are the same. Otherwise plotting follows the usual convention of image.
overlay	in the case of two objects x and y whether they should be plotted on top of one another (for pgrid objects the difference x-y is plotted) or not. In the presence of a transference plan overlay is forced to be true.
static.mass	for a transference plan that explicitly lists the "static mass transports" (i.e. mass that stays at the same site), should these transports also be plotted as disks with colours/sizes corresponding to the amount of mass that stays? Note that it is wrong to assume that an optimal transference plan obtained by one of the algorithms will automatically list static mass transports. It is not the case for $p = 1$, where static mass transport at site i is trivially equal to the minimum of source mass and target mass, and it is currently not the case for results obtained by method="aha".

Value

Used for its side effect.

Author(s)

Dominic Schuhmacher <dschuhm1@uni-goettingen.de>

---

plot.ut_pgrid *Plot Unbalanced Transport Information*

Description

Graphic representation of components of the list returned by unbalanced.

Usage

```
## S3 method for class 'ut_pgrid'
plot(x, what = c("plan", "extra", "trans", "inplace"), axes = FALSE, ...)
```

Arguments

x	the list returned by unbalanced with option output="all".
what	character. The aspect of the unbalanced transport information to display.
axes	logical. Whether to plot axes (ignored for what="plan").
...	further graphics parameters passed to plot.pgrid for what="plan" and passed to matimage in all other cases.

Value

Nothing. Used for the side effect.

Examples

```
## Not run:
res <- unbalanced( random32a, random32b, p=1, C=0.2, output="all" )
plot( res, what="plan", lwd=1.5, angle=20 )
plot( res, what="trans" )
plot( res, what="extra" )
plot( res, what="inplace" )
## End(Not run)
```

---

plot.ut_wpp *Plot Unbalanced Transport Information*

Description

Graphic representation of components of the list returned by unbalanced.

Usage

```
## S3 method for class 'ut_wpp'
plot(
  x,
  what = c("plan", "extra", "trans"),
  axes = FALSE,
  xlim = c(0, 1),
  ylim = c(0, 1),
  ...
)
```

Arguments

x	the list returned by unbalanced with option output="all".
what	character. The aspect of the unbalanced transport information to display.
axes	logical. Whether to plot axes (ignored for what="plan").
xlim, ylim	numeric vectors of length 2. The x- and y-limits of the plot.
...	further graphics parameters passed to plot.pgrid for what="plan" and passed to matimage in all other cases.

Value

Nothing. Used for the side effect.

Examples

```
## Not run:
set.seed(33)
m <- 50
n <- 20
massa <- rexp(m)
massb <- rexp(n)
a <- wpp( matrix(runif(2*m), m, 2), massa)
b <- wpp( matrix(runif(2*n), n, 2), massb)
res <- unbalanced(a,b,1,0.3,output="all")
plot(res, what="plan")
plot(res, what="trans")
plot(res, what="extra")
## End(Not run)
```

`plot_apollonius` *Plot Apollonius Diagram*

Description

Plots the Apollonius diagram, a.k.a. (additively) weighted Voronoi diagram, based on a matrix of points (centers) in 2d and their weights.

Usage

```
plot_apollonius(
  points,
  weights,
  show_points = TRUE,
  show_weights = TRUE,
  add_to_weights = 0,
  add = FALSE,
  col = 4,
  lwd = 1.5,
  ...
)
```

Arguments

`points`	A two-column matrix containing the 2d points.
`weights`	A vector of weights for the points.
`show_points`	Logical. Should the points be displayed in the plot? Defaults to TRUE.
`show_weights`	Logical. Should the weights be displayed in the plot? Defaults to TRUE.
`add_to_weights`	A value added to the weights to make the plot more informative.
`add`	Logical. Should the plot be added to the current device? Defaults to FALSE.
`col`	The colour for the cell boundaries.
`lwd`	The line width for the cell boundaries.
`...`	Further parameters to the base plot if `add` is `FALSE`.

Details

For points $x_1, \ldots, x_n$ with weights $w_1, \ldots, w_n$ The i-th cell of the Apollonius diagram contains all the points x that satisfy

$$\|x - x_i\| - w_i < \|x - x_j\| - w_j$$

for all $j \neq i$. Its boundaries are hyperbola segments.

If `show_weights` is `TRUE`, grey circles of radii `weights` + `add_to_weights` are plotted around the points. Negative radii are set to zero.

Note

This function requires the Computational Geometry Algorithms Library (CGAL), available at https://www.cgal.org. Adapt the file src/Makevars according to the instructions given there and re-install from source.

Author(s)

Valentin Hartmann <valentin.hartmann@epfl.ch> (most of the code)
Dominic Schuhmacher <schuhmacher@math.uni-goettingen.de> (R-port)

References

Menelaos Karavelas and Mariette Yvinec. 2D Apollonius Graphs (Delaunay Graphs of Disks). In CGAL User and Reference Manual. CGAL Editorial Board, 4.12 edition, 2018

Examples

```
## Not run:
w <- c(0.731, 0.0372, 0.618, 0.113, 0.395, 0.222, 0.124, 0.101, 0.328, 0)
points <- matrix(runif(20), 10, 2)
plot_apollonius(points, w, add_to_weights = -0.1)
## End(Not run)
```

`power_diagram` *Compute the Power Diagram of Weighted Sites in 2-Dimensional Space*

Description

Compute the power diagram of weighted sites in 2-dimensional space.

Usage

```
power_diagram(xi, eta, w, rect = NA)
## S3 method for class 'power_diagram'
plot(x, weights=FALSE, add=FALSE, col=4, lwd=1.5, ...)
```

Arguments

xi, eta, w	vectors of equal length, where xi, eta are the coordinates of the sites and w are the corresponding weights.
rect	vetor of length 4. To get a finite representation of the power diagram, it will be intersected with the rectangle $[rect[1], rect[2]] \times [rect[3], rect[4]]$. Defaults to c(min(xi),max(xi),min(eta),max(eta)).
x	a power diagram as returned from power_diagram.
weights	logical. If TRUE, weights of non-redundant sites with non-negative weight are represented as circles whose radii are equal to the square roots of the corresponding weights.
add	logical. Should the power diagram be plotted on top of current graphics?
col	the color of the cell boundaries.
lwd, ...	further arguments graphic parameters used by plot.default.

Details

The function power_diagram implements an algorithm by Edelsbrunner and Shah (1996) which computes regular triangulations and thus its dual representation, the power diagram. For point location, an algorithm devised by Devillers (2002) is used.

Author(s)

Björn Bähre <bjobae@gmail.com>
(slightly modified by Dominic Schuhmacher <dschuhm1@uni-goettingen.de>)

References

H. Edelsbrunner, N. R. Shah (1996), Incremental Topological Flipping Works for Regular Triangulations, Algorithmica 15, 223–241.

O. Devillers (2002), The Delaunay Hierarchy, International Journal of Foundations of Computer Science 13, 163–180.

Examples

```
xi <- runif(100)
eta <- runif(100)
w <- runif(100,0,0.005)
x <- power_diagram(xi,eta,w,rect=c(0,1,0,1))
plot(x,weights=TRUE)
```

pp *Constructor for the pp Class*

Description

Construct an object of class "pp" from a matrix.

Usage

```
pp(coordinates)
```

Arguments

coordinates	a matrix specifying the coordinates of the points. Each row corresponds to a point.

Details

For more detailed explanations of the arguments and other components of the derived object of class "pp", see pp-object.

Author(s)

Dominic Schuhmacher <dschuhm1@uni-goettingen.de>

See Also

Description of pp objects.

Examples

```
m <- matrix(c(1,1,2,2,3,1,4,2),4,2)
a <- pp(m)
print(a)
print.default(a)

## Not run:
  plot(a)
## End(Not run)
```

pp-object *Class of (Unweighted) Point Patterns*

Description

The class "pp" represents discrete measures with some fixed mass at any of finitely many locations.

Details

Objects of class "pp" may be created by the function pp, and are most commonly used as input to the function transport. There are methods plot, print and summary for this class.

An object of class "pp" contains the following elements:

dimension	the dimension of the Euclidean space in which the patterns live. Must be ≥ 2.
N	the total number of points.
coordinates	the coordinates of the points. An N $\times$ dimension matrix, where each row corresponds to a point.

Author(s)

Dominic Schuhmacher <dschuhm1@uni-goettingen.de>

See Also

Constructor function pp.

random *Images to Illustrate the Use of transport.pgrid*

Description

32 x 32, 64 x 64 and 128 x 128 images to illustrate the use of transport.pgrid. These are objects of class "pgrid".

Usage

```
random32a

random32b

random64a

random64b

random128a

random128b
```

Format

Objects of class 'pgrid'.

Source

Randomly generated using the package `RandomFields`.

`ret_message` *Return Text Strings for lbfgs Return Codes*

Description

Given a vector of return codes, give back the corresponding vector of return strings from the lbfgs library. Nonexistant codes are ignored.

Usage

```
ret_message(n = NULL)
```

Arguments

`n`	The vector of return codes or `NULL` meaning the whole list shall be returned.

Value

A named character vector of the corresponding return strings.

Note

Code 0 is ignored, since for technical reasons it is never returned by the function `semidiscrete1`.

Author(s)

Dominic Schuhmacher <schuhmacher@math.uni-goettingen.de>

See Also

`semidiscrete1`.

Examples

```
ret_message()
ret_message(c(2,-1023,-1019))
```

`semidiscrete` *Find Optimal Transport Partition Between pgrid and wpp.*

Description

Given an object `a` of class `pgrid` specifying an image and an object `b` of class `wpp` specifiying a more flexible mass distribution at finitely many points, find the partition of the image (and hence the optimal transport map) that minimizes the total transport cost for going from `a` to `b`.

Usage

```
semidiscrete(a, b, p = 2, method = c("aha"), control = list(), ...)
```

Arguments

`a`	an object of class `pgrid` usually representing an image or the discretization of a measure.
`b`	an object of class `wpp` usually having the same total mass as `a`.
`p`	the power ≥ 1 to which the Euclidean distance between points is taken in order to compute costs. Only $p \in \{1, 2\}$ is implemented.
`method`	the name of the algorithm to use. Currently only `aha` is supported.
`control`	a named list of parameters for the chosen method or the result of a call to `trcontrol`. Currently only the parameters `factr` and `maxit` can be set.
`...`	currently without effect.

Details

This is a wrapper for the functions `aha` and `semidiscrete1`. In the former the Aurenhammer–Hoffmann–Aronov (1998) method for $p = 2$ is implemented in the multiscale variant presented in Mérigot (2011). In the latter an adapted Aurenhammer–Hoffmann–Aronov method for $p = 1$ is used that was presented in Hartmann and Schuhmacher (2018).

The present function is automatically called by `transport` if the first argument is of class `pgrid` and the second argument is of class `wpp`.

Value

An object describing the optimal transport partition for `a` and `b`.

If `p=1` an object of class `apollonius_diagram` having components `sites` and `weights`, as well as (optionally) `wasserstein_dist` and `ret_code` (the return code from the call to `semidiscrete1`).

If `p=2` an objectof class `power_diagram` having components `sites` and `cells`, as well as (optionally) `wasserstein_dist`. `sites` is here a data.frame with columns `xi`, `eta` and `w` (the weights for the power diagram). `cells` is a list with as many 2-column matrix components as there are sites, each describing the x- and y-coordinates of the polygonal cell associated with the corresponding site or `NULL` if the cell of the site is empty.

Plotting methods exist for objects of class `apollonius_diagram`, `power_diagram` and for optimal transport maps represented by either of the two.

Note

For p=1 this function requires the Computational Geometry Algorithms Library (CGAL), available at https://www.cgal.org. Adapt the file src/Makevars according to the instructions given there and re-install from source.

Internally the code from liblbfgs 1.10 by Naoaki Okazaki (2010) is used.

Author(s)

Dominic Schuhmacher <dschuhm1@uni-goettingen.de>
Björn Bähre <bjobae@gmail.com>
Valentin Hartmann <valentin.hartmann@epfl.ch>

References

F. Aurenhammer, F. Hoffmann and B. Aronov (1998). Minkowski-type theorems and least-squares clustering. Algorithmica 20(1), 61–76.

V. Hartmann and D. Schuhmacher (2017). Semi-discrete optimal transport — the case p=1. Preprint arXiv:1706.07650

M. Karavelas and M. Yvinec. 2D Apollonius Graphs (Delaunay Graphs of Disks). In CGAL User and Reference Manual. CGAL Editorial Board, 4.12 edition, 2018

Q. Mérigot (2011). A multiscale approach to optimal transport. Computer Graphics Forum 30(5), 1583–1592. doi:10.1111/j.14678659.2011.02032.x

Naoaki Okazaki (2010). libLBFGS: a library of Limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS). Version 1.10

See Also

plot, transport, aha, semidiscrete1

Examples

```
##  See examples for function transport
```

---

semidiscrete1 *Compute Semidiscrete Optimal Transport for Euclidean Distance Cost*

Description

Computes the weight vector of the Apollonius diagram describing the semidiscrete optimal transport plan for the Euclidean distance cost function and the associated Wasserstein distance.

Usage

```
semidiscrete1(
  source,
  target,
  xrange = c(0, 1),
  yrange = c(0, 1),
  verbose = FALSE,
  reg = 0
)
```

Arguments

source	A matrix specifing the source measure.
target	A three-column matrix specifing the target measure in the form x-coordinate, y-coordinate, mass.
xrange, yrange	Vectors with two components defining the window on which the source measure lives. Defaults to $[0,1] \times [0,1]$. source is interpreted as an image of equally sized quadratic pixels on this window.
verbose	Logical. Shall information about multiscale progress and L-BFGS return codes be printed?
reg	A non-negative regularization parameter. It is usually not necessary to deviate from the default 0.

Value

A list describing the solution. The components are

weights	A vector of length equal to the first dimension of target containing the weights for the Apollonius diagram discribing the optimal semidiscrete transport from source to target.
wasserstein_dist	The L_1-Wasserstein distance between source and target.
ret_code	A return code. Equal to 1 if everything is OK, since our code interrupts the usual lbfgs code. Other values can be converted to the corresponding return message by using ret_message.

Note

This function requires the Computational Geometry Algorithms Library (CGAL), available at https://www.cgal.org. Adapt the file src/Makevars according to the instructions given there and re-install from source.

Internally the code from liblbfgs 1.10 by Naoaki Okazaki (2010) is used. See http://www.chokkan.org/software/liblbfgs/.

A stand-alone version of the C++ code of this function is available at https://github.com/valentin-hartmann-research/semi-discrete-transport.

Author(s)

Valentin Hartmann <valentin.hartmann@epfl.ch> (stand-alone C++ code)
Dominic Schuhmacher <schuhmacher@math.uni-goettingen.de> (R-port)

References

V. Hartmann and D. Schuhmacher (2017). Semi-discrete optimal transport — the case p=1. Preprint arXiv:1706.07650

Menelaos Karavelas and Mariette Yvinec. 2D Apollonius Graphs (Delaunay Graphs of Disks). In CGAL User and Reference Manual. CGAL Editorial Board, 4.12 edition, 2018

Naoaki Okazaki (2010). libLBFGS: a library of Limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS). Version 1.10

See Also

ret_message, semidiscrete.

Examples

```
## Not run:
# the following function rotates a matrix m clockwise, so
# that image(rococlock(m)) has the same orientation as print(m):
roclock <- function(m) t(m)[, nrow(m):1]

set.seed(30)
n <- 20
nu <- matrix(c(runif(2*n), rgamma(n,3,1)), n, 3)
pixelbdry <- seq(0,1,length=33)
image(pixelbdry, pixelbdry, roclock(random32a$mass), asp=1, col = grey(seq(0,1,length.out=32)))
points(nu[,1], nu[,2], pch=16, cex=sqrt(nu[,3])/2, col=2)

res <- semidiscrete1(random32a$mass, nu)
plot_apollonius(nu[,1:2], res$weights, show_weights = FALSE, add = TRUE)
points(nu[,1], nu[,2], pch=16, cex=sqrt(nu[,3])/2, col=2)
## End(Not run)
```

---

shielding — *Compute Optimal Transport (Cost/Plan) Using the Multiscale Shielding Method*

Description

Runs the multiscale version of the Shielding Method (a.k.a. Short Cut Method) for computing the optimal transport (cost/plan) on a rectangular grid in d dimensions for the squared Euclidean distance as cost function.

Usage

```
shielding(
  a,
  b,
  nscales = 2,
  startscale = 1,
  flood = 0,
  measureScale = 1e-06,
  verbose = FALSE,
  basisKeep = 1,
  basisRefine = 1
)
```

Arguments

a, b	arrays with d coordinates representing source and target measure, respectively. The entries must be all positive.
nscales	the number of scales generated in the multiscale algorithm.
startscale	the first scale on which the problem is solved.
flood	a real number. If positive, take the maximum of entry and flood for each entry of a and b.
measureScale	the required precision for the entries. Computations are performed on round(a/measureScale) and the same for b using integer arithmetics.
verbose	logical. Toggles output to the console about the progress of the algorithm.
basisKeep, basisRefine	internal use only.

Details

If a and b do not have the same sum, they are normalized to sum 1 *before* flood and measureScale transformations are applied.

Value

A list of components

err	error code. 0 if everything is ok.
a_used, b_used	the vectorized arrays that were actually used by the algorithm. a, b after applying flood and measureScale.
coupling	a vectorized coupling describing the optimal transport from a_used to b_used
basis	a matrix with two columns describing the basis obtained for the optimal transport
u, v	vectors of optimal values in the dual problem

Use of CPLEX

For larger problems (thousands of grid points) there are considerable speed improvements when `shielding` can use the CPLEX numerical solver for the underlying constrained optimization problems. If a local installation of CPLEX is available, the transport package can be linked against it during installation. See the file src/Makevars in the source package for instructions.

Author(s)

Bernhard Schmitzer <schmitzer@uni-muenster.de> and
Dominic Schuhmacher <dschuhm1@uni-goettingen.de>
(based on C++ code by Bernhard Schmitzer)

References

B. Schmitzer (2016). A sparse multiscale algorithm for dense optimal transport. J. Math. Imaging Vision 56(2), 238–259. https://arxiv.org/abs/1510.05466

See Also

`transport`, which calls this function if appropriate.

Examples

```
## Not run:
shielding(random64a$mass,random64b$mass,nscales=6,measureScale=1)
## End(Not run)
```

`starting solutions` *Compute starting solution for the transportation problem*

Description

Compute a feasible transference plan between two mass vectors.

Usage

```
northwestcorner(a, b)
russell(a, b, costm)
```

Arguments

`a, b`	Two numeric vectors (typically containing natural numbers) of length m and n, describing mass distributions.
`costm`	A m by n matrix of costs for moving one unit of mass.

Value

A list whose components are m by n matrices, viz.

`assignment`	containing as (i, j)-th entry the mass assigned from origin i to destination j;
`basis`	containing as (i, j)-th entry a 1 if it is a basic entry and a 0 otherwise.

Warnings

The current implementations are in R. Computations may be slow for larger vectors `a` and `b`.
The computed starting solution may be degenerate, i.e. there may be basic entries where zero mass is assigned.

Author(s)

Dominic Schuhmacher <dschuhm1@uni-goettingen.de>

See Also

`transport`

---

subwasserstein *Approximate Computation of Wasserstein Distances via Subsampling.*

Description

Samples `S` elements each of a source and a target measure and computes the Wasserstein distance between the samples. The mean distance out of `K` tries is returned.

Usage

```
subwasserstein(
  source,
  target,
  S,
  K = 1,
  p = 1,
  costM = NULL,
  prob = TRUE,
  precompute = FALSE,
  method = "networkflow"
)
```

Arguments

source	The source measure has to be either a weight vector or an object of one of the classes "pgrid", "wpp" or "pp".
target	The target measure needs to be of the same type as the source measure.
S	The sample size.
K	The number of tries. Defaults to 1.
p	The order of the Wasserstein metric (i.e. the power of the distances). Defaults to 1.
costM	The cost matrix between the source and target measures. Ignored unless source and target are weight vectors.
prob	logical. Should the objects a, b be interpreted as probability measures, i.e. their total mass be normalized to 1?
precompute	logical. Should the cost matrix for the large problem be precomputed?
method	A string with the name of the method used for optimal transport distance computation. Options are "revsimplex", "shortsimplex" and "primaldual". Defaults to "revsimplex".

Details

For larger problems setting precompute to TRUE is not recommended.

Value

The mean of the K values of the Wasserstein distances between the subsampled measures.

Author(s)

Jörn Schrieber <joern.schrieber-1@mathematik.uni-goettingen.de>
Dominic Schuhmacher <dominic.schuhmacher@mathematik.uni-goettingen.de>

References

M. Sommerfeld, J. Schrieber, Y. Zemel and A. Munk (2018) Optimal Transport: Fast Probabilistic Approximation with Exact Solvers preprint: arXiv:1802.05570

Examples

```
## Not run:
subwasserstein(random64a, random64b, S=1000)
wasserstein(random64a, random64b)

## End(Not run)
```

transport *Find Optimal Transport Plan Between Two Objects*

Description

Given two objects a and b that specify distributions of mass and an object that specifies (a way to compute) costs, find the transport plan for going from a to b that minimizes the total cost.

Usage

```
transport(a, b, ...)
## Default S3 method:
transport(a, b, costm, method = c("networkflow", "shortsimplex", "revsimplex",
"primaldual"), fullreturn=FALSE, control = list(), threads=1, ...)
## S3 method for class 'pgrid'
transport(a, b, p = NULL, method = c("auto", "networkflow", "revsimplex", "shortsimplex",
  "shielding", "aha", "primaldual"), fullreturn=FALSE,
  control = list(), threads=1,...)
## S3 method for class 'pp'
transport(a, b, p = 1, method = c("auction", "auctionbf", "networkflow", "shortsimplex",
 "revsimplex", "primaldual"), fullreturn=FALSE, control = list(), threads=1, ...)
## S3 method for class 'wpp'
transport(a, b, p = 1, method = c("networkflow", "revsimplex", "shortsimplex",
  "primaldual"), fullreturn=FALSE, control = list(), threads=1, ...)
```

Arguments

a, b	two objects that describe mass distributions, between which the optimal transport map is to be computed. For the default method these are vectors of non-negative values. For the other three methods these are objects of the respective classes. It is also possible to have a of class pgrid and b of class wpp.
costm	for the default method a length(a) by length(b) matrix specifying the cost of transporting single units of mass between the corresponding source and destination points.
p	for the three specialized methods the power ≥ 1 to which the Euclidean distance between points is taken in order to compute costs.
method	the name of the algorithm to use. See details below.
fullreturn	A boolean specifying whether the output of the function should also include the dual solution, the optimal transport cost between a and b and the transport plan in matrix form should be returned as well.
control	a named list of parameters for the chosen method or the result of a call to trcontrol. Any parameters that are not set by the control argument will get reasonable (sometimes problem specific) defaults.
threads	An Integer specifying the number of threads used in parallel computing. Currently only available for the method "networkflow".
...	currently without effect.

Details

There is a number of algorithms that are currently implemented and more will be added in future versions of the package. The following is a brief description of each key word used. Much more details can be found in the cited references and in a forthcoming package vignette.

`aha`: The Aurenhammer–Hoffmann–Aronov (1998) method with the multiscale approach presented in Mérigot (2011). The original theory was limited to $p = 2$. We refer by `aha` also to the extension of the same idea for $p = 1$ as presented in Hartmann and Schuhmacher (2017) and for more general p (currently not implemented).

`auction`: The auction algorithm by Bertsekas (1988) with epsilon-scaling, see Bertsekas (1992).

`auctionbf`: A refined auction algorithm that combines forward and revers auction, see Bertsekas (1992).

`networkflow`: The fast implementation of the network simplex algorithm by Nicolas Bonneel based on the LEMON Library (see citations below).

`primaldual`: The primal-dual algorithm as described in Luenberger (2003, Section 5.9).

`revsimplex`: The revised simplex algorithm as described in Luenberger and Ye (2008, Section 6.4) with various speed improvements, including a multiscale approach.

`shielding`: The shielding (or shortcut) method, as described in Schmitzer (2016).

`shortsimplex`: The shortlist method based an a revised simplex algorithm, as described in Gottschlich and Schuhmacher (2014).

The order of the *default* key words specified for the argument `method` gives a rough idea of the relative efficiency of the algorithms for the corresponding class of objects. For a given `a` and `b` the actual computation times may deviate significantly from this order. For class `pgrid` the default method is `"auto"`, which resolves to `"revsimplex"` if `p` is not 2 or the problem is very small, and to `"shielding"` otherwise.

The following table gives information about the applicability of the various algorithms (or sometimes rather their current implementations).

	default	pgrid	pp	wpp
aha (p=1 or 2!)	-	+	-	@
auction	-	-	+	-
auctionbf	-	-	+	-
networkflow	+	+	+	+
primaldual	*	*	*	+
revsimplex	+	+	*	+
shielding (p=2!)	-	+	-	-
shortsimplex	+	+	*	+

where: + recommended, * applicable (may be slow), - no implementation planned or combination does not make sense; @ indicates that the aha algorithm is available in the special combination where `a` is a `pgrid` object and `b` is a `wpp` object (and `p` is 2). For more details on this combination see the function `semidiscrete`.

Each algorithm has certain parameters supplied by the `control` argument. The following table gives an overview of parameter names and their applicability.

	start	multiscale	individual parameters
aha ($p = 2!$)	-	+	factr, maxit
auction	-	-	lasteps, epsfac
auctionbf	-	-	lasteps, epsfac
networkflow	-	-	
primaldual	-	-	
revsimplex	+	+	
shielding ($p = 2!$)	-	+	
shortsimplex	-	-	slength, kfound, psearched

start specifies the algorithm for computing a starting solution (if needed). Currently the Modified Row Minimum Rule (start="modrowmin"), the North-West Corner Rule (start="nwcorner") and the method by Russell (1969) (start="russell") are implemented. When start="auto" (the default) the ModRowMin Rule is chosen. However, for transport.pgrid and p larger than 1, there are two cases where an automatic multiscale procedure is also performed, i.e. the optimal transport is first computed on coarser grids and information from these solutions is then used for the finer girds. This happens for method = "revsimplex", where a single coarsening at factor scmult=2 is performed, and for method = "shielding", where a number of coarsenings adapted to the dimensions of the array is performed.

For p=1 and method="revsimplex", as well as p=2 and method="aha" there are multiscale versions of the corresponding algorithms that allows for finer control via the parameters nscales, scmult and returncoarse. The default value of nscales=1 suppresses the multiscale version. For larger problems it is advisable to use the multiscale version, which currently is only implemented for square pgrids in two dimensions. The algorithm proceeds then by coarsening the pgrid nscales-1 times, summarizing each time scmult^2 pixels into one larger pixels, and then solving the various transport problems starting from the coarsest and using each previous problem to compute a starting solution to the next finer problem. If returncoarse is TRUE, the coarser problems and their solutions are returned as well (revsimplex only).

factr, maxit are the corresponding components of the control argument in the optim L-BFGS-B method.

lasteps, epsfac are parameters used for epsilon scaling in the auction algortihm. The algorithm starts with a "transaction cost" per bid of epsfac^k * lasteps for some reasonable k generating finer and finer approximate solutions as the k counts down to zero. Note that in order for the procedure to make sense, epsfac should be larger than one (typically two- to three-digit) and in order for the final solution to be exact lasteps should be smaller than 1/n, where n is the total number of points in either of the point patterns. slength, kfound, psearched are the shortlist length, the number of pivot candidates needed, and the percentage of shortlists searched, respectively.

Value

A data frame with columns from, to and mass that specifies from which element of a to which element of b what amount of mass is sent in the optimal transport plan. For class pgrid elements are specified as vector indices in terms of the usual column major enumeration of the matrices a$mass and b$mass. There are plot methods for the classes pgrid and pp, which can plot this solution.

If returncoarse is TRUE for the revsimplex method, a list with components sol and prob giving the solutions and problems on the various scales considered. The solution on the finest scale (i.e. the output we obtain when setting returncoarse to FALSE) is in sol[[1]].

If a is of class pgrid and b of class wpp (and p=2), an object of class power_diagram as described in the help for the function semidiscrete. The plot method for class pgrid can plot this solution.

Use of CPLEX

The combination of the shielding-method with the CPLEX numerical solver outperforms the other algorithms by an order of magnitude for large problems (only applicable for p=2 and objects of class "pgrid"). If a local installation of CPLEX is available, the transport package can be linked against it during installation. See the file src/Makevars in the source package for instructions.

Use of CGAL

The combination of the aha-method with p=1 requires the use of CGAL (the Computational Geometry Algorithms Library) for dealing with Apollonius diagrams. If you require this functionality, install it from https://www.cgal.org/download.html and adapt the file src/Makevars of this package according to the instructions given in that file. Then re-install 'transport' from source as usual.

Author(s)

Dominic Schuhmacher <schuhmacher@math.uni-goettingen.de>

Björn Bähre <bjobae@gmail.com> (code for aha-method for p=2)

Nicolas Bonneel <nicolas.bonneel@liris.cnrs.fr>
(adaption of LEMON code for fast networkflow method)

Carsten Gottschlich <gottschlich@math.uni-goettingen.de>
(original java code for shortlist and revsimplex methods)

Valentin Hartmann <valentin.hartmann@epfl.ch> (code for aha method for p=1)

Florian Heinemann <florian.heinemann@uni-goettingen.de>
(integration of networkflow method)

Bernhard Schmitzer <schmitzer@uni-muenster.de> (code for shielding-method)

References

F. Aurenhammer, F. Hoffmann and B. Aronov (1998). Minkowski-type theorems and least-squares clustering. Algorithmica 20(1), 61–76.

D. P. Bertsekas (1988). The auction algorithm: a distributed relaxation method for the assignment problem. Annals of Operations Research 14(1), 105–123.

D. P. Bertsekas (1992). Auction algorithms for network flow problems: a tutorial introduction. Computational Optimization and Applications 1, 7–66.

N. Bonneel (2018). Fast Network Simplex for Optimal Transport. Github repository, nbonneel/network_simplex.

N. Bonneel, M. van de Panne, S. Paris and W. Heidrich (2011). Displacement interpolation using Lagrangian mass transport. ACM Transactions on Graphics (SIGGRAPH ASIA 2011) 30(6).

Egervary Research Group on Combinatorial Optimization, EGRES (2014). LEMON Graph Library v1.3.1. lemon.cs.elte.hu/trac/lemon.

C. Gottschlich and D. Schuhmacher (2014). The shortlist method for fast computation of the earth mover's distance and finding optimal solutions to transportation problems. PLOS ONE 9(10), e110214. doi:10.1371/journal.pone.0110214

V. Hartmann and D. Schuhmacher (2020). Semi-discrete optimal transport: a solution procedure for the unsquared Euclidean distance case, Mathematical Methods of Operations Research 92, 133–163. doi:10.1007/s0018602000703z

D.G. Luenberger (2003). Linear and nonlinear programming, 2nd ed. Kluwer.

D.G. Luenberger and Y. Ye (2008). Linear and nonlinear programming, 3rd ed. Springer.

Q. Mérigot (2011). A multiscale approach to optimal transport. Computer Graphics Forum 30(5), 1583–1592. doi:10.1111/j.14678659.2011.02032.x

B. Schmitzer (2016). A sparse multiscale algorithm for dense optimal transport. J. Math. Imaging Vision 56(2), 238–259. https://arxiv.org/abs/1510.05466

See Also

plot, wasserstein, unbalanced.

Examples

```
#
# example for the default method
#
a <- c(100, 200, 80, 150, 50, 140, 170, 30, 10, 70)
b <- c(60, 120, 150, 110, 40, 90, 160, 120, 70, 80)
set.seed(24)
costm <- matrix(sample(1:20, 100, replace=TRUE), 10, 10)
res <- transport(a,b,costm)

# pretty-print solution in matrix form for very small problems:
transp <- matrix(0,10,10)
transp[cbind(res$from,res$to)] <- res$mass
rownames(transp) <- paste(ifelse(nchar(a)==2," ",""),a,sep="")
colnames(transp) <- paste(ifelse(nchar(b)==2," ",""),b,sep="")
print(transp)


#
# example for class 'pgrid'
#
dev.new(width=9, height=4.5)
par(mfrow=c(1,2), mai=rep(0.1,4))
image(random32a$mass, col = grey(0:200/200), axes=FALSE)
image(random32b$mass, col = grey(0:200/200), axes=FALSE)
res <- transport(random32a,random32b)
dev.new()
par(mai=rep(0,4))
plot(random32a,random32b,res,lwd=1)


#
```

```
# example for class 'pp'
#
set.seed(27)
x <- pp(matrix(runif(400),200,2))
y <- pp(matrix(runif(400),200,2))
res <- transport(x,y)
dev.new()
par(mai=rep(0.02,4))
plot(x,y,res)


#
# example for class 'wpp'
#
set.seed(30)
m <- 30
n <- 60
massx <- rexp(m)
massx <- massx/sum(massx)
massy <- rexp(n)
massy <- massy/sum(massy)
x <- wpp(matrix(runif(2*m),m,2),massx)
y <- wpp(matrix(runif(2*n),n,2),massy)
res <- transport(x,y,method="revsimplex")
plot(x,y,res)


#
# example for semidiscrete transport between class
# 'pgrid' and class 'wpp' (p=2)
#
set.seed(33)
n <- 100
massb <- rexp(n)
massb <- massb/sum(massb)*1e5
b <- wpp(matrix(runif(2*n),n,2),massb)
res <- transport(random32a,b,p=2)
plot(random32a,b,res)


#
# example for semidiscrete transport between class
# 'pgrid' and class 'wpp' (p=1)
#
if (transport:::cgal_present()) {
  set.seed(33)
  n <- 30
  massb <- rexp(n)
  massb <- massb/sum(massb)*1e5
  b <- wpp(matrix(runif(2*n),n,2),massb)
  res <- transport(random32a,b,p=1)
  plot(random32a,b,res)
}
```

transport_track *Create a Dynamic Visualization of a Transference Plan Between Two pgrids*

Description

Given two objects source and target of class pgrid and a transference plan, typically the result of a call to transport, create an animation of the dynamic transference plan (a.k.a. displacement interpolation)

Usage

```
transport_track(source, target, tplan, K = 50, scmult = 1, smooth = FALSE,
   H = matrix(c(1,0,0,1),2,2), create.file = c("none","gif_im"),
   file.name = "Rtransport.gif", fps = 20, cut = FALSE,
   col=grey((0:1000)/1000),width=800,height=800)
```

Arguments

source, target	objects of class pgrid.
tplan	a transference plan between source and target, typically an optimal transference plan obtained by a call to transport.
K	the number of intermediate frames to be produced between source and target.
scmult	the factor by which the number of pixels in each dimension is multiplied to obtain a smoother rendering of the dynamic transference plan.
smooth	logical. Whether a kernel smoothing or a linear binning procedure is used to generate the images. Defaults to FALSE.
H	the bandwith matrix used to perform the two dimensional kernel density estimation or the linear binning respectively.
create.file	the file type to be created or "none" to return only an array of intermediate mass distributions.
file.name	the path for the output file. Ignored if create.file is "none".
fps	the number of frames per second in the generated gif. The default is 20 frames per second.
cut	logical. Whether the boundary pixels are cut off. Currently the only way to deal with the edge effect (see Details).
col	the vector of RGB colours which is used to generate the gif, if create.file is not "none". See the documentation of image for more details.
width	interger specifying the width of the images used to generate the output gif, if create.file is not "none".
height	interger specifying the width of the images used to generate the output gif, if create.file is not "none".

Details

The intermediate frames are produced by the interpolation formula $[(1-t)\mathrm{pr}_1 + t\mathrm{pr}_2]_{\#}\pi$, where π is the transference plan, pr_1 and pr_2 are the first and second coordinate projections of $\mathbf{R}^2 \times \mathbf{R}^2$ onto $\mathbf{R}^2$, and $t \in \{0, 1/(K+1), \ldots, K/(K+1), 1\}$. If π is an optimal transference plan this yields the displacement interpolation, at least if we assume as underlying cost function the Euclidean metric to the p-th power, where $p = 1, 2$.

The kernel smoothing procedure gives usually nicer animations, but takes several orders of magnitudes longer.

There are currently visible edge effects in both the kernel smoothing and the linear binning procedure that lead to darker pixels at the boundary of the image. The cut parameter may be used to remove the boundary pixels completely and thus produce a smaller output. The edge will be dealt with more adequatly in future versions.

Conversion to an animated gif is performed by a system call to the convert tool of ImageMagick. The latter may have to be installed first.

Value

An array containing the various interpolation images.

Unless `create.file="none"`, the function is mainly used for its side effect (saving a file to the specified path). So the array is returned invisibly.

Warning

Running this function with `smooth=TRUE` and even moderate `K` can take a long time!

Author(s)

Florian Heinemann <florian.heinemann@uni-goettingen.de>
(slightly modified by Dominic Schuhmacher <dschuhm1@uni-goettingen.de>)

See Also

Function `transport` for computing optimal transference plans.

Examples

```
if (requireNamespace("ks", quietly = TRUE)) {
  tplan <- transport(random32a,random32b)
  series <- transport_track(random32a, random32b, tplan, scmult=3, create.file="none")
  dev.new(width=16,height=8)
  oldpar <- par(mfrow=c(5,10), mai=rep(0.01,4))
  for (i in 1:50) {
   image(series[,,i], col=grey(seq(0,1,0.005)), asp=1, axes=FALSE,zlim=c(min(series),max(series)))
  }
  par(oldpar)
}
```

trcontrol *Set the Control Parameters Used by* transport.

Description

Set the control parameters for the algorithm used by the function transport.

Usage

```
trcontrol(method = c("networkflow", "revsimplex", "shortsimplex", "primaldual", "aha",
          "shielding", "auction", "auctionbf"), para = list(),
          start = c("auto",          "modrowmin", "nwcorner", "russell"),
          nscales = 1, scmult = 2,returncoarse = FALSE, a = NULL,
          b = NULL, M = NULL, N = NULL)
```

Arguments

method	The algorithm to be used to compute the optimal transference plan. See details for the function transport.pgrid.
para	A list of parameters that are specific to the chosen method. See the table on the help page of the function transport.
start	If method == "revsimplex", the method for computing a starting solution.
nscales, scmult, returncoarse	The parameters for the multiscale versions of certain algorithms. See the help on transport.
a, b, M, N	The two objects a and b for which the transportation problem is to be solved *or* the sizes M and N of these objects. Based on the information available here, trcontrol tries hard to find reasonable values for the control parameters of the algorithm not specified directly.

Details

For further details about the parameters of the individual algorithms see the help page for transport.

Value

A list with components method, para, start, nscales, scmult, returncoarse as entered or adapted/computed based on the arguments method, a, b, M, N.

Note

This function is typically only called by the user to check what the parameter settings used by the function transport are for a given problem.

Author(s)

Dominic Schuhmacher <dschuhm1@uni-goettingen.de>

See Also

transport

---

unbalanced *Unbalanced Optimal Transport Between Two Objects*

Description

Compute optimal transport between unnormalized images / mass distributions on grids (pgrid objects) or between mass distributions on general point patterns (wpp objects) under the option that mass can be dispose of. Transport cost per unit is the Euclidean distance of the transport to the p-th power. Disposal cost per unit is C^p.

Usage

```
unbalanced(a, b, ...)

## S3 method for class 'pgrid'
unbalanced(
  a,
  b,
  p = 1,
  C = NULL,
  method = c("networkflow", "revsimplex"),
  output = c("dist", "all", "rawres"),
  threads = 1,
  ...
)

## S3 method for class 'wpp'
unbalanced(
  a,
  b,
  p = 1,
  C = NULL,
  method = c("networkflow", "revsimplex"),
  output = c("dist", "all", "rawres"),
  threads = 1,
  ...
)
```

Arguments

a, b	objects of class pgrid or wpp that are compatible.
...	other arguments.

p	a power ≥ 1 applied to the transport and disposal costs. The order of the resulting unbalanced Wasserstein metric.
C	The base disposal cost (without the power p)
method	one of "networkflow" and "revsimplex", specifing the algorithm used. See details.
output	character. One of "dist", "all" and "rawres". Determines what the function returns: only the unbalanced Wasserstein distance; all available information about the transport plan and the extra mass; or the raw result obtained by the networkflow algorithm. The latter is the same format as in the transport function with option fullreturn=TRUE. The choice output = "rawres" is mainly intended for internal use.
threads	an integer specifying the number of threads for parallel computing in connection with the networkflow method.

Details

Given two non-negative mass distributions $a = (a_x)_{x\in S}$, $b = (a_y)_{y\in S}$ on a set S (a pixel grid / image if a, b are of class pgrid or a more general weighted point pattern if a, b are of class wpp), this function minimizes the functional

$$\sum_{x,y\in S} \pi_{x,y} d(x,y)^p + C^p\big(\sum_{x\in S}(a_x - \pi_x^{(1)}) + \sum_{y\in S}(b_y - \pi_y^{(2)})\big)$$

over all $(\pi_{x,y})_{x,y\in S}$ satisfying

$$0 \leq \pi_x^{(1)} := \sum_{y\in S} \pi_{x,y} \leq a_x \text{ and } 0 \leq \pi_y^{(2)} := \sum_{x\in S} \pi_{x,y} \leq b_y.$$

Thus $\pi_{x,y}$ denotes the amount of mass transported from x to y, whereas $\pi_x^{(1)}$ and $\pi_y^{(2)}$ are the total mass transported away from x and total mass transported to y, respectively. Accordingly $\sum_{x\in S}(a_x - \pi_x^{(1)})$ and $\sum_{y\in S}(b_y - \pi_y^{(2)})$ are the total amounts of mass of a and b, respectively, that need to be disposed of.

The minimal value of the functional above taken to the $1/p$ is what we refer to as unbalanced (p, C)-Wasserstein metric. This metric is used, in various variants, in an number of research papers. See Heinemann et al. (2022) and the references therein and Müller et al. (2022), Remark 3. We follow the convention of the latter paper regarding the parametrization and the use of the term *unbalanced Wasserstein metric*.

The practical difference between the two methods "networkflow" and "revsimplex" can roughly described as follows. The former is typically faster for large examples (for pgrid objects 64x64 and beyond), especially if several threads are used. The latter is typically faster for smaller examples (which may be relevant if pairwise transports between many objects are computed) and it guarantees a sparse(r) solution, i.e. at most $m + n + 1$ individual transports, where m and n are the numbers of non-zero masses in a and b, respectively). Note however that due to the implementation the revsimplex algorithm is a little less precise (roughly within 1e-7 tolerance). For more details on the algorithms see transport.

Value

If `output = "dist"` a single numeric, the unbalanced (p, C)-Wasserstein distance. Otherwise a list. If `output = "all"` the list is of class `ut_pgrid` or `ut_wpp` according to the class of the objects `a` and `b`. It has `a`, `b`, `p`, `C` as attributes and the following components:

`dist`	same as for `output = "dist"`.
`plan`	an optimal transport plan. This is a data frame with columns `from`, `to` and `mass` that specifies from which element of `a` to which element of `b` what amount of mass is sent. `from` and `to` are specified as vector indices in terms of the usual column major enumeration of the matrices a$mass and b$mass. The plan can be plotted via `plot.pgrid(a, b, plan)`.
`atrans`, `btrans`	matrices (pgrid) or vectors (wpp) specifying the masses transported from each point and to each point, respectively. Corresponds to $(\pi_x^{(1)})_{x\in S}$ and $(\pi_y^{(2)})_{y\in S}$ above.
`aextra`, `bextra`	matrices (pgrid) or vectors (wpp) specifying the amount of mass at each point of `a` and `b`, respectively, that cannot be transported and needs to be disposed of. Corresponds to $(a_x - \pi_x^{(1)})_{x\in S}$ and $(b_y - \pi_y^{(2)})_{y\in S}$.
`inplace`	(pgrid only) a matrix specifying the amount of mass at each point that can stay in place. Corresponds to $(\pi_{x,x})_{x\in S}$.

Note that `atrans + aextra + inplace` (pgrid) or `atrans + aextra` (wpp)must be equal to `a$mass` and likewise for `b`. A warning occurs if this is not the case (which may indeed happen from time to time for method revsimplex, but the error reported should be very small).

References

Florian Heinemann, Marcel Klatt and Axel Munk (2022).
Kantorovich-Rubinstein distance and barycenter for finitely supported measures: Foundations and Algorithms.
Arxiv preprint.
doi:10.48550/arXiv.2112.03581

Raoul Müller, Dominic Schuhmacher and Jorge Mateu (2020).
Metrics and barycenters for point pattern data Statistics and Computing 30, 953-972.
doi:10.1007/s1122202009932y

See Also

`plot.ut_pgrid` and `plot.ut_wpp`, which can plot the various components of the list obtained for `output="all"`.

Examples

```
a <- pgrid(matrix(1:12, 3, 4))
b <- pgrid(matrix(c(9:4, 12:7), 3, 4))
res1 <- unbalanced(a, b, 1, 0.5, output="all")
res2 <- unbalanced(a, b, 1, 0.3, output="all")
plot(a, b, res1$plan, angle=20, rot=TRUE)
```

```
plot(a, b, res2$plan, angle=20, rot=TRUE)
par(mfrow=c(1,2))
matimage(res2$aextra, x = a$generator[[1]], y = a$generator[[2]])
matimage(res2$bextra, x = b$generator[[1]], y = b$generator[[2]])

set.seed(31)
a <- wpp(matrix(runif(8),4,2), 3:6)
b <- wpp(matrix(runif(10),5,2), 1:5)
res1 <- unbalanced(a, b, 1, 0.5, output="all")
res2 <- unbalanced(a, b, 1, 0.3, output="all")
plot(a, b, res1$plan)
plot(a, b, res2$plan)
```

`wasserstein` *Compute the Wasserstein Distance Between Two Objects*

Description

Given two objects a and b that specify measures in R^d, compute the Wasserstein distance of order p between the objects.

Usage

```
wasserstein(a, b, p=1, tplan=NULL, costm=NULL, prob=TRUE, ...)
```

Arguments

a, b	two objects that describe mass distributions in R^d. Either both of class pgrid or pp or wpp or numeric. For the first three the dimension d of the structures must be at least 2; see function wasserstein1d for $d = 1$.
p	the power ≥ 1 to which the Euclidean distance between points is taken in order to compute transportation costs.
tplan	an optional transference plan in the format returned by the function transport. If NULL an optimal transference plan based on a, b and p is computed by a call to transport.
costm	the matrix of costs between the support points of the measures. Ignored unless a and b are numeric vectors.
prob	logical. Should the objects a, b be interpreted as probability measures, i.e. their total mass be normalized to 1?
...	further parameters passed to transport if tplan is NULL.

Details

The Wasserstein distance of order p is defined as the p-th root of the total cost incurred when transporting measure a to measure b in an optimal way, where the cost of transporting a unit of mass from x to y is given as the p-th power $\|x-y\|^p$ of the Euclidean distance.

If tplan is supplied by the user, no checks are performed whether it is optimal for the given problem. So this function may be used to compare different (maybe suboptimal) transference plans with regard to their total costs.

For further details on the algorithms used, see help of transport.

Value

A single number, the Wasserstein distance for the specified problem.

Author(s)

Dominic Schuhmacher <dschuhm1@uni-goettingen.de>

See Also

plot, transport, wasserstein1d

Examples

```
#
# example for class 'pgrid'
#
wasserstein(random32a,random32b,p=1)
res <- transport(random32a,random32b,p=2)
wasserstein(random32a,random32b,p=1,res)
# is larger than above:
# the optimal transport for p=2 is not optimal for p=1

#
# example for class 'pp'
#
set.seed(27)
x <- pp(matrix(runif(500),250,2))
y <- pp(matrix(runif(500),250,2))
wasserstein(x,y,p=1)
wasserstein(x,y,p=2)
```

---

wasserstein1d *Compute the Wasserstein Distance Between Two Univariate Samples*

Description

Given two vectors a and b, compute the Wasserstein distance of order p between their empirical distributions.

Usage

```
wasserstein1d(a, b, p = 1, wa = NULL, wb = NULL)
```

Arguments

a, b	two vectors.
p	a positive number. The order of the Wasserstein distance.
wa, wb	optional vectors of non-negative weights for a and b.

Details

The Wasserstein distance of order p is defined as the p-th root of the total cost incurred when transporting a pile of mass into another pile of mass in an optimal way, where the cost of transporting a unit of mass from x to y is given as the p-th power $\|x-y\|^p$ of the Euclidean distance.

In the present function the vector a represents the locations on the real line of m deposits of mass $1/m$ and the vector b the locations of n deposits of mass $1/n$. If the user specifies weights wa and wb, these default masses are replaced by wa/sum(wa) and wb/sum(wb), respectively.

In terms of the empirical distribution function $F(t) = \sum_{i=1}^{m} w_i^{(a)} 1\{a_i \le t\}$ of locations a_i with normalized weights $w_i^{(a)}$, and the corresponding function $G(t) = \sum_{j=1}^{n} w_j^{(b)} 1\{b_j \le t\}$ for b, the Wasserstein distance in 1-d is given as

$$W_p(F,G) = \left(\int_0^1 |F^{-1}(u) - G^{-1}(u)|^p \, du \right)^{1/p},$$

where F^{-1} and G^{-1} are generalized inverses. If $p = 1$, we also have

$$W_1(F,G) = \int_{-\infty}^{\infty} |F(x) - G(x)| \, dx.$$

Value

A single number, the Wasserstein distance for the specified data.

Author(s)

Dominic Schuhmacher <dschuhm1@uni-goettingen.de>

See Also

wasserstein

Examples

```
x <- rnorm(200)
y <- rnorm(150,2)
wasserstein1d(x,y)
```

wpp *Constructor for the wpp Class*

Description

Construct an object of class "wpp" from a matrix of points and a vector of masses.

Usage

```
wpp(coordinates, mass)
```

Arguments

coordinates	a matrix specifying the coordinates of the points. Each row corresponds to a point.
mass	a vector of non-negative values specifying the masses at these points.

Details

For more detailed explanations of the arguments and other components of the returned object of class "wpp", see wpp-object.

It is legitimate to assign mass 0 to individual points in the arguments. However, when constructing the wpp-object such points are deleted. The coordinates of the deleted points can still be accessed via the attribute zeropoints.

Author(s)

Dominic Schuhmacher <dschuhm1@uni-goettingen.de>
Timo Wilm <timo.wilm@stud.uni-goettingen.de>

See Also

Description of pp objects.

Examples

```
m <- matrix(c(1,1,2,2,3,1,4,2),4,2)
a <- pp(m)
print(a)
print.default(a)

## Not run:
  plot(a)
## End(Not run)
```

wpp-object *Class of Weighted Point Patterns*

Description

The class "wpp" represents discrete measures with positive mass at any of finitely many locations.

Details

Objects of class "wpp" may be created by the function wpp, and are most commonly used as input to the function transport. There are methods plot, print and summary for this class.

An object of class "wpp" contains the following elements:

dimension	the dimension of the Euclidean space in which the patterns live. Must be ≥ 2.
N	the total number of point.
coordinates	the coordinates of the points. An N $\times$ dimension matrix, where each row corresponds to a point.
mass	the masses at these points. A vector of length N of positive numbers.
totmass	the total mass of the point pattern.

Author(s)

Dominic Schuhmacher <dschuhm1@uni-goettingen.de>
Timo Wilm <timo.wilm@stud.uni-goettingen.de>

See Also

Constructor function wpp.

Index

∗ **aha**
transport, 28
∗ **auctionbf**
transport, 28
∗ **auction**
transport, 28
∗ **code**
ret_message, 19
∗ **control**
trcontrol, 36
∗ **datasets**
random, 18
∗ **earth mover**
wasserstein, 40
wasserstein1d, 41
∗ **grid**
pgrid, 9
pgrid-object, 10
∗ **main function**
transport, 28
∗ **mallows**
wasserstein, 40
wasserstein1d, 41
∗ **networkflow**
transport, 28
∗ **object**
pgrid, 9
pgrid-object, 10
pp, 17
pp-object, 18
wpp, 43
wpp-object, 44
∗ **package**
transport-package, 3
∗ **point pattern**
pp, 17
pp-object, 18
wpp-object, 44
∗ **primaldual**
transport, 28
∗ **return**
ret_message, 19
∗ **revsimplex**
transport, 28
∗ **semi-discrete**
semidiscrete, 20
∗ **shielding**
transport, 28
∗ **shortsimplex**
transport, 28
∗ **transport parameters**
trcontrol, 36
∗ **transport partition**
semidiscrete, 20
∗ **transport tesselation**
semidiscrete, 20
∗ **weighted point pattern**
wpp, 43
∗ **weighted**
wpp-object, 44
∗ **weight**
wpp-object, 44

aha, *3*, 4, *20, 21*, *31*
all.equal, *7*
all.equal (transport objects), 6
all.equal.pgrid *(*all.equal (transport objects)*)*, 6
all.equal.pp *(*all.equal (transport objects)*)*, 6
all.equal.wpp *(*all.equal (transport objects)*)*, 6

compatible, 6, *6*

image, *7*, *10*, *12*

matimage, 7, *12, 13*
methods, 8

mode, *6*

networkflow, *3*
northwestcorner *(*starting solutions*)*, 25

optim, *4*, *30*
optimal transport maps represented by either of the two, *20*

pgrid, *3*, *5*, 9, *10*, *20*, *37*, *40*
pgrid objects, *9*
pgrid-object, 10
plot, *3*, *5*, *10*, 11, *18*, *21*, *30–32*, *41*, *44*
plot.default, *16*
plot.pgrid, *12, 13*
plot.power_diagram *(*power_diagram*)*, 15
plot.ut_pgrid, 12, *39*
plot.ut_wpp, 13, *39*
plot_apollonius, 14
power_diagram, *5*, 15, *16*
pp, *3*, *5*, 17, *18*, *40*
pp objects, *17*, *43*
pp-object, 18
print, *10*, *18*, *44*
print.pgrid *(*methods*)*, 8
print.pp *(*methods*)*, 8
print.wpp *(*methods*)*, 8

random, 18
random128 *(*random*)*, 18
random128a *(*random*)*, 18
random128b *(*random*)*, 18
random32 *(*random*)*, 18
random32a *(*random*)*, 18
random32b *(*random*)*, 18
random64 *(*random*)*, 18
random64a *(*random*)*, 18
random64b *(*random*)*, 18
ret_message, 19, *22, 23*
revsimplex, *3*
russell *(*starting solutions*)*, 25

semidiscrete, 20, *23*, *29*, *31*
semidiscrete1, *19, 20*, 21, *21*
shielding, *3*, 23, *31*
shortlist, *3*
starting solutions, 25
subsampling, *3*
subwasserstein, 26

summary, *10*, *18*, *44*
summary.pgrid *(*methods*)*, 8
summary.pp *(*methods*)*, 8
summary.wpp *(*methods*)*, 8

transport, *3–6*, *10, 11*, *18*, *20, 21*, *25, 26*, 28, *34–38*, *40, 41*, *44*
transport-package, 3
transport.pgrid, *18*, *36*
transport_apply *(*aha*)*, 4
transport_error *(*aha*)*, 4
transport_track, 34
trcontrol, *20*, *28*, 36

unbalanced, *12, 13*, *32*, 37

wasserstein, *3*, *32*, 40, *42*
wasserstein1d, *3*, *40*, 41, *41*
wpp, *3*, *20*, *37*, *40*, 43, *44*
wpp-object, 44