Package ‘sphericalDepth’

December 6, 2024

Type Package

Title Computation of the spherical halfspace depth

Version 0.9.2

Author Rainer Dyckerhoff

Maintainer Rainer Dyckerhoff <rainer.dyckerhoff@statistik.uni-koeln.de>

Description
This package provides two routines to compute the spherical halfspace depth, also known as angular halfspace depth or angular Tukey depth. For a description of the algorithms, see: Dyckerhoff, R., Nagy, S. (2024) Exact computation of angular halfspace depth.

License GPL-2

Encoding UTF-8

LazyData true

RoxygenNote 7.2.3

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

URL https://github.com/DyckerhoffRainer/sphericalDepth

BugReports https://github.com/DyckerhoffRainer/sphericalDepth/issues

R topics documented:

`ahD` *Angular halfspace depth*

Description

This function computes the angular halfspace depth (it is not assumed that the points are in general position).

Usage

```
ahD(
  x,
  mass = NULL,
  z = NULL,
  ind = if (missing(z)) 1:nrow(as.matrix(x)) else NULL,
  alg = c("comb", "rec"),
  target = 2,
  method = c("standard", "adaptive", "single", "multiple", "nGP", "GP"),
  nThreads = 0
)
```

Arguments

`x`	a matrix of format `n`-times-`d` that contains the data points with respect to which the depth has to be computed
`mass`	a vector of length `n` that contains either the probabilities or the (integer) multiplicities of the `n` data points
`z`	a matrix of format `m`-times-`d` containing the points whose depth should be computed
`ind`	a vector of length `l` containing the indices of the data points `x` whose depth should be computed
`alg`	a string, possible values are `"comb"` and `"rec"`. Denotes the algorithm that is used to compute the signed halfspace depth in the target dimension. If `alg="comb"`, the combinatorial algorithm is used to compute the angular halfspace depth. If `alg="rec"` the recursive algorithm is used. For details, see Dyckerhoff and Nagy (2024).
`target`	the dimension to which the data is projected, possible values are `1`, `2`, or `3`
`method`	a string, possible values are `"standard"`, `"adaptive"`, `"single"`, `"multiple"`, `"nGP"`, `"GP"`. Denotes the variant of the algorithm that is used to compute the signed halfspace depth in the target dimension. For `target=2`, only `"standard"`, `"adaptive"`, `"single"`, `"multiple"` are valid. For `target=2`, `method="standard"` is the same as `method="adaptive"`, meaning that depending on the number of points for which the depth has to be computed, either `"single"` or `"multiple"` is selected. For `target=3`, only `"standard"`, `"nGP"`, `"GP"` are valid. For `target=3`, `method="standard"` is the same as `method="nGP"`, meaning that the points do not have to be in general position. If `method="GP"`, then it is assumed that all points are in general position, so that a faster algorithm can be used.
`nThreads`	the number of threads to use in a multiprocessor environment. For the default `0`, the number of threads is automatically chosen depending on the used hardware.

Details

Unless `target=3` and `method="GP"`, the algorithm does not assume that the points are in general position. The points in `z` may coincide with points in `x`. `z` or `ind` may be missing. If both `z` and `ind` are missing, thenthe depths of all points in `x` are calculated.

Regarding the choice of parameters `alg`, `target`, and `method`, the default values usually will give the best results. In most cases the combinatorial algorithm (`alg="comb"`) called with the default values for `target` and `method` will give the best results.

Value

a vector of length `m+l` containing the angular halfspace depths of the points `z` and the data points `x` whose indices are given in `ind`. If the argument `mass` is missing or if an integer vector is supplied for `mass`, then the depths are given as integer values, i.e., multiplied by `n`. If a double vector is supplied for the argument `mass`, then the depths are given as doubles.

Author(s)

Rainer Dyckerhoff

References

Dyckerhoff, R., and Nagy, S. (2024). Exact computation of angular halfspace depth.

Examples

```
d <- 4
n <- 50
m <- 50
l <- 50
# simulate data uniformly distributed on the sphere
x <- matrix(rnorm(n*d),nrow=n)
x <- x / sqrt(rowSums(x*x))
z <- matrix(rnorm(m*d),nrow=m)
z <- z / sqrt(rowSums(z*z))
# vector of probabilities
prob <- rep(1/n, n)
# vector of point multiplicities
count <- as.integer(rep(1,n))

res <- matrix(nrow=30, ncol=m+l)

# combinatorial algorithm, w/o argument 'mass', different target dimensions and methods
res[ 1,] <- ahD(x, z = z, ind = 1:l, alg = "comb", target = 1)
res[ 2,] <- ahD(x, z = z, ind = 1:l, alg = "comb", target = 2, method = "single")
res[ 3,] <- ahD(x, z = z, ind = 1:l, alg = "comb", target = 2, method = "multiple")
res[ 4,] <- ahD(x, z = z, ind = 1:l, alg = "comb", target = 3, method = "nGP")
res[ 5,] <- ahD(x, z = z, ind = 1:l, alg = "comb", target = 3, method = "GP")

# combinatorial algorithm, pass a vector of probabilities to mass, different target dimensions
# and methods
res[ 6,] <- ahD(x, mass = prob, z = z, ind = 1:l, alg = "comb", target = 1)
res[ 7,] <- ahD(x, mass = prob, z = z, ind = 1:l, alg = "comb", target = 2, method = "single")
res[ 8,] <- ahD(x, mass = prob, z = z, ind = 1:l, alg = "comb", target = 2, method = "multiple")
res[ 9,] <- ahD(x, mass = prob, z = z, ind = 1:l, alg = "comb", target = 3, method = "nGP")
res[10,] <- ahD(x, mass = prob, z = z, ind = 1:l, alg = "comb", target = 3, method = "GP")
# multiply by n since the depths are since we want to compare with the integer version of the depth
res[6:10,] <- res[6:10,] * n

# combinatorial algorithm, pass a vector of multiplicities to mass, different target dimensions
# and methods
res[11,] <- ahD(x, mass = count, z = z, ind = 1:l, alg = "comb", target = 1)
res[12,] <- ahD(x, mass = count, z = z, ind = 1:l, alg = "comb", target = 2, method = "single")
res[13,] <- ahD(x, mass = count, z = z, ind = 1:l, alg = "comb", target = 2, method = "multiple")
res[14,] <- ahD(x, mass = count, z = z, ind = 1:l, alg = "comb", target = 3, method = "nGP")
res[15,] <- ahD(x, mass = count, z = z, ind = 1:l, alg = "comb", target = 3, method = "GP")
```

```
# recursive algorithm, w/o argument 'mass', different target dimensions and methods
res[16,] <- ahD(x, z = z, ind = 1:l, alg = "rec", target = 1)
res[17,] <- ahD(x, z = z, ind = 1:l, alg = "rec", target = 2, method = "single")
res[18,] <- ahD(x, z = z, ind = 1:l, alg = "rec", target = 2, method = "multiple")
res[19,] <- ahD(x, z = z, ind = 1:l, alg = "rec", target = 3, method = "nGP")
res[20,] <- ahD(x, z = z, ind = 1:l, alg = "rec", target = 3, method = "GP")

# recursive algorithm, pass a vector of probabilities to mass, different target dimensions
# and methods
res[21,] <- ahD(x, mass = prob, z = z, ind = 1:l, alg = "rec", target = 1)
res[22,] <- ahD(x, mass = prob, z = z, ind = 1:l, alg = "rec", target = 2, method = "single")
res[23,] <- ahD(x, mass = prob, z = z, ind = 1:l, alg = "rec", target = 2, method = "multiple")
res[24,] <- ahD(x, mass = prob, z = z, ind = 1:l, alg = "rec", target = 3, method = "nGP")
res[25,] <- ahD(x, mass = prob, z = z, ind = 1:l, alg = "rec", target = 3, method = "GP")
# multiply by n since the depths are since we want to compare with the integer version of the depth
res[21:25,] <- res[21:25,] * n

# recursive algorithm, pass a vector of multiplicities to mass, different target dimensions
# and methods
res[26,] <- ahD(x, mass = count, z = z, ind = 1:l, alg = "rec", target = 1)
res[27,] <- ahD(x, mass = count, z = z, ind = 1:l, alg = "rec", target = 2, method = "single")
res[28,] <- ahD(x, mass = count, z = z, ind = 1:l, alg = "rec", target = 2, method = "multiple")
res[29,] <- ahD(x, mass = count, z = z, ind = 1:l, alg = "rec", target = 3, method = "nGP")
res[30,] <- ahD(x, mass = count, z = z, ind = 1:l, alg = "rec", target = 3, method = "GP")

print(paste("Number of different results: ",sum(apply(res, 2, max) - apply(res, 2, min) > 1e-13)))
```

---

ahD_Comb *Angular halfspace depth*

Description

This function computes the angular halfspace depth using the combinatorial algorithm (it is not assumed that the points are in general position). ahD_Comb is deprecated and should no longer be used. Applications should use ahD instead.

Usage

```
ahD_Comb(
  x,
  mass = NULL,
  z = NULL,
  ind = NULL,
  target = 2,
  method = c("standard", "adaptive", "single", "multiple", "nGP", "GP"),
  nThreads = 0
)
```

Arguments

x	a matrix of format n-times-d that contains the data points with respect to which the depth has to be computed

mass	a vector of length n that contains either the probabilities or the (integer) multiplicities of the n data points
z	a matrix of format m-times-d containing the points whose depth should be computed
ind	a vector of length l containing the indices of the data points x whose depth should be computed
target	the dimension to which the data is projected, possible values are 1, 2, or 3
method	a string, possible values are "standard", "adaptive", "single", "multiple", "nGP", "GP". Denotes the variant of the algorithm that is used to compute the signed halfspace depth in the target dimension. For target=2, only "standard", "adaptive", "single", "multiple" are valid. For target=2, method="standard" is the same as method="adaptive", meaning that depending on the number of points for which the depth has to be computed, either "single" or "multiple" is selected. For target=3, only "standard", "nGP", "GP" are valid. For target=3, method= "standard" is the same as method="nGP", meaning that the points do not have to be in general position. If method="GP", then it is assumed that all points are in general position, so that a faster algorithm can be used.
nThreads	the number of threads to use in a multiprocessor environment. For the default 0, the number of threads is automatically chosen depending on the used hardware.

Details

The routine uses the combinatorial algorithm of Dyckerhoff and Nagy (2024). Unless target=3 and method="GP", it does not assume that the points are in general position. The points in z may coincide with points in x. z or ind may be missing (but not both).

Regarding the choice of parameters target and method, the default values usually will give the best results. In most cases this routine (ahD_Comb) called with the default values for target and method will give the best results and will also be faster than ahD_Rec.

Value

a vector of length m+l containing the angular halfspace depths of the points z and the data points x whose indices are given in ind. If the argument mass is missing or if an integer vector is supplied for mass, then the depths are given as integer values, i.e., multiplied by n. If a double vector is supplied for the argument mass, then the depths are given as doubles.

Author(s)

Rainer Dyckerhoff

References

Dyckerhoff, R., and Nagy, S. (2024). Exact computation of angular halfspace depth.

Examples

```
d <- 4
n <- 50
m <- 50
l <- 50
# simulate data uniformly distributed on the sphere
```

```
x <- matrix(rnorm(n*d),nrow=n)
x <- x / sqrt(rowSums(x*x))
z <- matrix(rnorm(m*d),nrow=m)
z <- z / sqrt(rowSums(z*z))
# vector of probabilities
prob <- rep(1/n, n)
# vector of point multiplicities
count <- as.integer(rep(1,n))

res <- matrix(nrow=20, ncol=m+l)

# combinatorial algorithm, w/o argument 'mass', different target dimensions and methods
res[ 1,] <- ahD_Comb(x, z = z, ind = 1:l, target = 1)
res[ 2,] <- ahD_Comb(x, z = z, ind = 1:l, target = 2, method = "single")
res[ 3,] <- ahD_Comb(x, z = z, ind = 1:l, target = 2, method = "multiple")
res[ 4,] <- ahD_Comb(x, z = z, ind = 1:l, target = 3, method = "nGP")
res[ 5,] <- ahD_Comb(x, z = z, ind = 1:l, target = 3, method = "GP")

# combinatorial algorithm, pass a vector of probabilities to mass, different target dimensions
# and methods
res[ 6,] <- ahD_Comb(x, mass = prob, z = z, ind = 1:l, target = 1)
res[ 7,] <- ahD_Comb(x, mass = prob, z = z, ind = 1:l, target = 2, method = "single")
res[ 8,] <- ahD_Comb(x, mass = prob, z = z, ind = 1:l, target = 2, method = "multiple")
res[ 9,] <- ahD_Comb(x, mass = prob, z = z, ind = 1:l, target = 3, method = "nGP")
res[10,] <- ahD_Comb(x, mass = prob, z = z, ind = 1:l, target = 3, method = "GP")
# multiply by n since the depths are since we want to compare with the integer version of the depth
res[6:10,] <- res[6:10,] * n

# combinatorial algorithm, pass a vector of multiplicities to mass, different target dimensions
# and methods
res[11,] <- ahD_Comb(x, mass = count, z = z, ind = 1:l, target = 1)
res[12,] <- ahD_Comb(x, mass = count, z = z, ind = 1:l, target = 2, method = "single")
res[13,] <- ahD_Comb(x, mass = count, z = z, ind = 1:l, target = 2, method = "multiple")
res[14,] <- ahD_Comb(x, mass = count, z = z, ind = 1:l, target = 3, method = "nGP")
res[15,] <- ahD_Comb(x, mass = count, z = z, ind = 1:l, target = 3, method = "GP")

# recursive algorithm, different target dimensions and methods
res[16,] <- ahD_Rec(x, z = z, ind = 1:l, target = 1)
res[17,] <- ahD_Rec(x, z = z, ind = 1:l, target = 2, method = "single")
res[18,] <- ahD_Rec(x, z = z, ind = 1:l, target = 2, method = "multiple")
res[19,] <- ahD_Rec(x, z = z, ind = 1:l, target = 3, method = "nGP")
res[20,] <- ahD_Rec(x, z = z, ind = 1:l, target = 3, method = "GP")

print(paste("Number of different results:  ",sum(apply(res, 2, max) - apply(res, 2, min) > 1e-13)))
```

ahD_Rec *Angular halfspace depth*

Description

This function computes the angular halfspace depth using the recursive algorithm (it is not assumed that the points are in general position). `ahD_Rec` is deprecated and should no longer be used. Applications should use `ahD` instead.

Usage

```
ahD_Rec(
  x,
  mass = NULL,
  z = NULL,
  ind = NULL,
  target = 2,
  method = c("standard", "adaptive", "single", "multiple", "nGP", "GP"),
  nThreads = 0
)
```

Arguments

`x`	a matrix of format `n`-times-`d` that contains the data points with respect to which the depth has to be computed
`mass`	a vector of length `n` that contains either the probabilities or the (integer) multiplicities of the `n` data points
`z`	a matrix of format `m`-times-`d` containing the points whose depth should be computed
`ind`	a vector of length `l` containing the indices of the data points `x` whose depth should be computed
`target`	the dimension to which the data is projected, possible values are `1`, `2`, or `3`
`method`	a string, possible values are `"standard"`, `"adaptive"`, `"single"`, `"multiple"`, `"nGP"`, `"GP"`. Denotes the variant of the algorithm that is used to compute the signed halfspace depth in the target dimension. For `target=2`, only `"standard"`, `"adaptive"`, `"single"`, `"multiple"` are valid. For `target=2`, `method="standard"` is the same as `method="adaptive"`, meaning that depending on the number of points for which the depth has to be computed, either `"single"` or `"multiple"` is selected. For `target=3`, only `"standard"`, `"nGP"`, `"GP"` are valid. For `target=3`, `method= "standard"` is the same as `method="nGP"`, meaning that the points do not have to be in general position. If `method="GP"`, then it is assumed that all points are in general position, so that a faster algorithm can be used.
`nThreads`	the number of threads to use in a multiprocessor environment. For the default `0`, the number of threads is automatically chosen depending on the used hardware.

Details

The routine uses the recursive algorithm of Dyckerhoff and Nagy (2024). Unless `target=3` and `method="GP"`, it does not assume that the points are in general position. The points in `z` may coincide with points in x. z or ind may be missing (but not both).

Regarding the choice of parameters `target`, and `method`, the default values usually will give the best results. In most cases this routine will be slower than 'ahD_Comb'.

Value

a vector of length `m+l` containing the angular halfspace depths of the points `z` and the data points `x` whose indices are given in `ind`. If the argument `mass` is missing or if an integer vector is supplied for `mass`, then the depths are given as integer values, i.e., multiplied by `n`. If a double vector is supplied for the argument `mass`, then the depths are given as doubles.

Author(s)

Rainer Dyckerhoff

References

Dyckerhoff, R., and Nagy, S. (2024). Exact computation of angular halfspace depth.

Examples

```
d <- 4
n <- 50
m <- 50
l <- 50
# simulate data uniformly distributed on the sphere
x <- matrix(rnorm(n*d),nrow=n)
x <- x / sqrt(rowSums(x*x))
z <- matrix(rnorm(m*d),nrow=m)
z <- z / sqrt(rowSums(z*z))
# vector of probabilities
prob <- rep(1/n, n)
# vector of point multiplicities
count <- as.integer(rep(1,n))

res <- matrix(nrow=20, ncol=m+l)

# recursive algorithm, w/o argument 'mass', different target dimensions and methods
res[ 1,] <- ahD_Rec(x, z = z, ind = 1:l, target = 1)
res[ 2,] <- ahD_Rec(x, z = z, ind = 1:l, target = 2, method = "single")
res[ 3,] <- ahD_Rec(x, z = z, ind = 1:l, target = 2, method = "multiple")
res[ 4,] <- ahD_Rec(x, z = z, ind = 1:l, target = 3, method = "nGP")
res[ 5,] <- ahD_Rec(x, z = z, ind = 1:l, target = 3, method = "GP")

# recursive algorithm, pass a vector of probabilitiues to mass, different target dimensions
#and methods
res[ 6,] <- ahD_Rec(x, mass = prob, z = z, ind = 1:l, target = 1)
res[ 7,] <- ahD_Rec(x, mass = prob, z = z, ind = 1:l, target = 2, method = "single")
res[ 8,] <- ahD_Rec(x, mass = prob, z = z, ind = 1:l, target = 2, method = "multiple")
res[ 9,] <- ahD_Rec(x, mass = prob, z = z, ind = 1:l, target = 3, method = "nGP")
res[10,] <- ahD_Rec(x, mass = prob, z = z, ind = 1:l, target = 3, method = "GP")
# multiply by n since the depths are since we want to compare with the integer version of the depth
res[6:10,] <- res[6:10,] * n

# recursive algorithm, pass a vector of multiplicieties to mass, different target dimensions
# and methods
res[11,] <- ahD_Rec(x, mass = count, z = z, ind = 1:l, target = 1)
res[12,] <- ahD_Rec(x, mass = count, z = z, ind = 1:l, target = 2, method = "single")
res[13,] <- ahD_Rec(x, mass = count, z = z, ind = 1:l, target = 2, method = "multiple")
res[14,] <- ahD_Rec(x, mass = count, z = z, ind = 1:l, target = 3, method = "nGP")
res[15,] <- ahD_Rec(x, mass = count, z = z, ind = 1:l, target = 3, method = "GP")

# combinatorial algorithm, different target dimensions and methods
res[16,] <- ahD_Comb(x, z = z, ind = 1:l, target = 1)
res[17,] <- ahD_Comb(x, z = z, ind = 1:l, target = 2, method = "single")
res[18,] <- ahD_Comb(x, z = z, ind = 1:l, target = 2, method = "multiple")
res[19,] <- ahD_Comb(x, z = z, ind = 1:l, target = 3, method = "nGP")
res[20,] <- ahD_Comb(x, z = z, ind = 1:l, target = 3, method = "GP")
```

```
print(paste("Number of different results:  ",sum(apply(res, 2, max) - apply(res, 2, min) > 1e-13)))
```

Index