Package ‘alphashape’

May 1, 2020

Type Package

Title Create Delaunay triangulations, Voronoi vertices and alpha shape for n number of dimension using the QHULL library

Version 1.2

Date 2019-03-15

Maintainer Pascal Omondiagbe <omondiagbep@landcareresearch.co.nz>

Description Makes an Alpha shape for any number of N dimension using the Delaunay triangulations generated using via the Qhull library (www.qhull.org)
This package has been tested to work up to 5 number of dimension.

License MIT + file LICENSE

Depends R (>= 3.5.1)

Imports devtools

Encoding UTF-8

LazyData true

RoxygenNote 7.1.0

Suggests testthat

NeedsCompilation no

R topics documented:

alphashape 2
alpha_complex 3
alpha_shape 4
convex_hull 5
convex_layer 6
delaunay 7
find_simplex 8
grid_coordinates 9
in_convex_hull 9
voronoi 10

Index **12**

`alphashape` *Computation of n dimension α-shape*

Description

Implementation in n dimension of the alpha shape using the Q-hull library

Package:	alphashape
Date:	2019-03-14
License:	GPL-2
LazyLoad:	yes

Author(s)

Pascal Omondiagbe
Tom Etherington
Maintainers: pascal Omondiagbe <omondiagbep@landcareresearch.co.nz>

References

http://www.qhull.org/html/qh-code.htm

`alpha_complex` *Alpha complex*

Description

This function calculates the alpha complex of a set of n points in d-dimensional space using the Qhull library.

Usage

```
alpha_complex(points = NULL, alpha = Inf)
```

Arguments

`points` a n-by-d dataframe or matrix. The rows represent n points and the d columns the coordinates in d-dimensional space.

`alpha` a real number between zero and infinity that defines the maximum circumradii for a simplex to be included in the alpha complex. If unspecified `alpha` defaults to infinity.

Value

Returns a list consisting of: [1] a s-by-$d+1$ matrix of point indices that define the s simplices that make up the alpha complex; [2] a s-by-d matrix of circumcentres for each simplex ; [3] a list of s circumradii for each simplex; and [4] the input points used to create the alpha complex.

References

Barber CB, Dobkin DP, Huhdanpaa H (1996) The Quickhull algorithm for convex hulls. ACM Transactions on Mathematical Software, 22(4):469-83 https://doi.org/10.1145/235815.235821.

Edelsbrunner H, Mücke EP (1994) Three-dimensional alpha shapes. ACM Transactions on Graphics, 13(1):43-72 https://dl.acm.org/doi/abs/10.1145/174462.156635.

Examples

```
# Define points
x <- c(30, 70, 20, 50, 40, 70)
y <- c(35, 80, 70, 50, 60, 20)
p <- data.frame(x, y)
# Create alpha complex and plot
a_complex <- alpha_complex(points = p, alpha = 20)
plot(p, pch = as.character(seq(nrow(p))), xlim=c(0,80), ylim=c(10,90), asp=1)
for (s in seq(nrow(a_complex$simplices))) {
  polygon(a_complex$input_points[a_complex$simplices[s,],], border="red")
}
text(a_complex$circumcentres, labels=seq(nrow(a_complex$simplices)), col="blue")
symbols(a_complex$circumcentres, circles = a_complex$circumradii,
        inches = FALSE, add = TRUE, fg="blue", lty="dotted")
```

`alpha_shape` *alpha_shape*

Description

Compute an alpha Shape Grid using the Q-hull library.

Usage

```
alpha_shape(simplicies, alpha_range, maxs, mins, n)
```

Arguments

`simplicies`	A Delaunay trigulation list object created by `delaunay` or a alpha complex list object created by `alpha_complex`.
`alpha_range,`	range of alpha value
`mins`	Vector of length `n` listing the point space minimum for each dimension. @param maxs Vector of length `n` listing the point space maximum for each dimension.
`n`	`n` dimension point co-ordinate

Details

The calculation is done by assigning the trigulation index when the grid cell center lies within the trigulation or -1 if it lies outside

Value

grid stack as vector, gridSimplex, and the inputed grid point.

Examples

```
# Define points and create a Delaunay triangulation
x <- c(30, 70, 20, 50, 40, 70)
y <- c(35, 80, 70, 50, 60, 20)
p <- data.frame(x, y)
a_complex <- alpha_complex(points = p, alpha = 20)
alpha_shape(simplicies = a_complex,maxs = c(70,80),mins = c(20,20),n = 5,alpha_range = c(1:20))
```

---

convex_hull *Convex hull*

Description

This function calculates the convex hull around a set of n points in d-dimensional space using the Qhull library.

Usage

```
convex_hull(points = NULL)
```

Arguments

points	a n-by-d dataframe or matrix. The rows represent n points and the d columns the coordinates in d-dimensional space.

Value

Returns a list consisting of: [1] a matrix for which each row is the pair of point indices that define the egde of the convex hull; [2] a vector of the point indicies that form the convex hull; [3] a matrix of point coordinates that form the convex hull; and [4] the input points used to create the convex hull.

References

Barber CB, Dobkin DP, Huhdanpaa H (1996) The Quickhull algorithm for convex hulls. ACM Transactions on Mathematical Software, 22(4):469-83 https://doi.org/10.1145/235815.235821.

See Also

convex_layer

Examples

```
# Define points
x <- c(30, 70, 20, 50, 40, 70)
y <- c(35, 80, 70, 50, 60, 20)
p <- data.frame(x, y)
# Create convex hull and plot
ch <- convex_hull(points = p)
plot(p, pch = as.character(seq(nrow(p))))
for (e in seq(nrow(ch$hull_edges))) {
  lines(ch$input_points[ch$hull_edges[e, ], ], col = "red")
}
```

convex_layer *Convex layer*

Description

This function calculates a convex layer of specified depth from a set of n points in d-dimensional space using the Qhull library.

Usage

```
convex_layer(points = NULL, layer = 1)
```

Arguments

points	a n-by-d dataframe or matrix. The rows represent n points and the d columns the coordinates in d-dimensional space.
layer	an integer that specifies the desired convex layer. If left unspecified convex layer 1 is returned that is equivalent to the convex hull.

Value

Returns a list consisting of: [1] a matrix for which each row is the pair of point indices that define the egde of the convex layer; [2] a vector of the point indicies that form the convex layer; [3] a matrix of point coordinates that form the convex layer; and [4] the input points used to create the convex layer.

References

Barber CB, Dobkin DP, Huhdanpaa H (1996) The Quickhull algorithm for convex hulls. ACM Transactions on Mathematical Software, 22(4):469-83 https://doi.org/10.1145/235815.235821.

See Also

convex_hull

Examples

```
# Create some random example data
set.seed(1) # to reproduce figure exactly
x = 20 + rgamma(n = 100, shape = 3, scale = 2)
y = rnorm(n = 100, mean = 280, sd = 30)
p <- data.frame(x, y)
plot(p)
cols <- c("red", "blue", "orange", "lightseagreen", "purple")
for (i in seq(5)) {
  cl <- convex_layer(points = p, layer = i)
  for (e in seq(nrow(cl$hull_edges))) {
    lines(cl$input_points[cl$hull_edges[e, ], ], col = cols[i], lwd = 2)
  }
}
legend("topright", legend = seq(5), lwd = 2, col = cols, bty = "n",
       title = "Convex layers")
```

delaunay *Delaunay triangulation*

Description

This function calculates the Delaunay triangulation of a set of n points in d-dimensional space using the Qhull library.

Usage

```
delaunay(points = NULL)
```

Arguments

points	a n-by-d dataframe or matrix. The rows represent n points and the d columns the coordinates in d-dimensional space.

Value

Returns a list consisting of: [1] a s-by-$d+1$ matrix of point indices that define the s simplices that make up the Delaunay triangulation; [2] a list containing for each simplex the neighbouring simplices; and [3] the input points used to create the Delaunay triangulation.

References

Barber CB, Dobkin DP, Huhdanpaa H (1996) The Quickhull algorithm for convex hulls. ACM Transactions on Mathematical Software, 22(4):469-83 https://doi.org/10.1145/235815.235821.

Examples

```
# Define points
x <- c(30, 70, 20, 50, 40, 70)
y <- c(35, 80, 70, 50, 60, 20)
p <- data.frame(x, y)
# Create Delaunay triangulation and plot
dt <- delaunay(points = p)
plot(p, pch = as.character(seq(nrow(p))))
for (s in seq(nrow(dt$simplices))) {
  polygon(dt$input_points[dt$simplices[s,],], border="red")
  text(x=colMeans(dt$input_points[dt$simplices[s,],])[1],
       y=colMeans(dt$input_points[dt$simplices[s,],])[2],
       labels=s, col="red")
}
```

`find_simplex` *Find simplex*

Description

Returns the simplicies of a Delaunay triangulation or alpha complex that contain the given set of test points.

Usage

```
find_simplex(simplicies, test_points)
```

Arguments

`simplicies`	A Delaunay trigulation list object created by `delaunay` or a alpha complex list object created by `alpha_complex` that contain simplicies.
`test_points`	a n-by-d dataframe or matrix. The rows represent n points and the d columns the coordinates in d-dimensional space.

Value

A n length vector containing the index of the simplex the test point is within, or a value of NA if a test point is not within any of the simplicies.

Examples

```
# Define points and create a Delaunay triangulation
x <- c(30, 70, 20, 50, 40, 70)
y <- c(35, 80, 70, 50, 60, 20)
p <- data.frame(x, y)
a_complex <- alpha_complex(points = p, alpha = 20)
# Check which simplex the test points belong to
p_test <- data.frame(c(20, 50, 60, 40), c(20, 60, 60, 50))
p_test_simplex <- find_simplex(simplicies = a_complex, test_points = p_test)
plot(p, pch = as.character(seq(nrow(p))), xlim=c(0,90))
for (s in seq(nrow(a_complex$simplices))) {
  polygon(a_complex$input_points[a_complex$simplices[s,],], border="red")
  text(x=colMeans(a_complex$input_points[a_complex$simplices[s,],])[1],
       y=colMeans(a_complex$input_points[a_complex$simplices[s,],])[2],
       labels=s, col="red")
}
points(p_test[,1], p_test[,2], pch=c("1", "2", "3", "4"), col="blue")
legend("topright", legend = c("input points", "simplicies", "test points"),
       text.col=c("black", "red", "blue"), title = "Indicies for:", bty="n")
print(p_test_simplex)
```

`grid_coordinates` *Grid Coordinates*

Description

Create an n-dimensional grid of coordinates across space.

Usage

```
grid_coordinates(mins, maxs, nCoords)
```

Arguments

`mins`	Vector of length `n` listing the point space minimum for each dimension.
`maxs`	Vector of length `n` listing the pointspace maximum for each dimension.
`nCoords`	Number of coordinates across the point space in all dimensions.

Details

This function creates a grid of coordinates systematically located throughout the specified point space to enable visualisation of alpha shape . The extent of the grid is given by the `mins` and `maxs`, and the number of coordinates for each dimension is given by `nCoords`.

Value

A matrix with `n` columns.

Examples

```
# Point space grid coordinates usage
xy = grid_coordinates(mins=c(15,0), maxs=c(35,200), nCoords=5)
```

`in_convex_hull` *In convex hull*

Description

Given a d-dimensional convex hull this function checks to see which of a set of n test points are within the convex hull. This function uses the Qhull library.

Usage

```
in_convex_hull(hull = NULL, test_points = NULL)
```

Arguments

`hull`	A convex hull list object created by `convex_hull`
`test_points`	a n-by-d dataframe or matrix. The rows represent n points and the d columns the coordinates in d-dimensional space.

Value

A n length vector containing TRUE if test point n lies within the hull and FALSE if it lies outside the hull.

References

Barber CB, Dobkin DP, Huhdanpaa H (1996) The Quickhull algorithm for convex hulls. ACM Transactions on Mathematical Software, 22(4):469-83 https://doi.org/10.1145/235815.235821.

See Also

convex_hull

Examples

```
# Define points to create the convex hull
x <- c(30, 70, 20, 50, 40, 70)
y <- c(35, 80, 70, 50, 60, 20)
p <- data.frame(x, y)
ch <- convex_hull(points = p)
# Check if some test points are in the convex hull
p_test <- data.frame(c(20, 50, 60), c(20, 50, 60))
checks <- in_convex_hull(hull = ch, test_points = p_test)
```

---

voronoi *Voronoi diagram*

Description

This function calculates the Voronoi digram of a set of n points in d-dimensional space using the Qhull library.

Usage

```
voronoi(points = NULL)
```

Arguments

points	a n-by-d dataframe or matrix. The rows represent n points and the d columns the coordinates in d-dimensional space.

Value

Returns a list consisting of...

References

Barber CB, Dobkin DP, Huhdanpaa H (1996) The Quickhull algorithm for convex hulls. ACM Transactions on Mathematical Software, 22(4):469-83 https://doi.org/10.1145/235815.235821.

Examples

```
# Define points
x <- c(30, 70, 20, 50, 40, 70)
y <- c(35, 80, 70, 50, 60, 20)
p <- data.frame(x, y)
# Create Voronoi diagram and plot
vd <- voronoi(points = p)
plot(p, pch = as.character(seq(nrow(p))))
```

Index

∗Topic **package**
alphashape, 2

alpha_complex, 3, *4*, *8*
alpha_shape, 4
alphashape, 2

convex_hull, 5, *6*, *9*, *10*
convex_layer, *5*, 6

delaunay, *4*, 7, *8*

find_simplex, 8

grid_coordinates, 9

in_convex_hull, 9

voronoi, 10