

# Package ‘alphashape’

April 29, 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](http://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** 7.0.2

**Suggests** testthat

**NeedsCompilation** no

## R topics documented:

alphashape . . . . .	1
alpha_complex . . . . .	2
alpha_shape . . . . .	3
convex_hull . . . . .	4
delaunay . . . . .	5
find_simplex . . . . .	6
grid_coordinates . . . . .	6
in_convex_hull . . . . .	7
voronoi . . . . .	8
<b>Index</b>	<b>9</b>

---

 alphashape

*Computation of  $n$  dimension  $\alpha$ -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 = 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.

**alpha** a positive number that defines the maximum circumradii for a simplex to be included in the alpha complex.

### Value

Returns a list consisting of: [1] a  $n$ -by- $d + 1$  matrix of point indices that define the **simplices** that make up the alpha complex; [2] a list circumcentres for each simplex ; [3] a list of 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>.

## 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))))
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")
}
```

---

alpha\_shape

*alpha\_shape*


---

## Description

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

## Usage

```
alpha_shape(point, alphaRange, maxs, mins, n)
```

## Arguments

point	observation as dataframe or matrix
alphaRange,	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 inputted grid point.

## Examples

```
x = c(30,70,20,50,40,70,20)
y = c(35,80,70,50,60,20,30)
p = data.frame(x,y)
alpha_shape(point = p,maxs = c(70,80),mins = c(20,20),n = 5,alphaRange = c(1:20))
```

---

convex_hull	<i>Convex hull</i>
-------------	--------------------

---

## 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 edge of the convex hull; [2] a vector of the point indices 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))))
polygon(ch$hull_points, border = "red")
```

---

delaunay	<i>Delaunay triangulation</i>
----------	-------------------------------

---

## 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  $n$ -by- $d + 1$  matrix of point indices that define the **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	<i>Find Simplex</i>
--------------	---------------------

---

### Description

Returns the simplicies of the delaunay trigulation which contains a given point.

### Usage

```
find_simplex(tri, inputPoint, testPoint)
```

### Arguments

tri	delaunay trigulation simplex using <a href="#">delaunay</a>
inputPoint	n-by-dim dataframe or matrix of original inputPoint.
testPoint	n-by-dim dataframe of points or matrix to check.

### Details

Given a grid point and a test point point, the find Simplex will identify the simplicies contianing the test point. It works by first checking for all point inside a convex hull, and then check if the center of the grid cell is inside the trigulation

### Value

A n\*m vector containing the result. -1 if a given point is outside the trigulation, or the trigulation index if the center of the cell is inside the alpha shape

### Examples

```
x = c(30,70,20,50,40,70)
y = c(35,80,70,50,60,20)
p = data.frame(x,y)
v=voronoi(point =p)
meshGrdiSpace = grid_coordinates(mins=c(15,0), maxs=c(35,200), nCoords=5)
simplex <- find_simplex(v$tri, v$inputPoints,meshGrdiSpace)
```

---

grid_coordinates	<i>Grid Coordinates</i>
------------------	-------------------------

---

### 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	<i>In convex hull</i>
----------------	-----------------------

---

**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 <a href="#">convex_hull</a>
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 new points are in the convex hull
new <- data.frame(c(20, 50), c(20, 50))
checks <- in_convex_hull(hull = ch, test_points = new)
```

---

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, [1](#)

alpha\_complex, [2](#)

alpha\_shape, [3](#)

alphashape, [1](#)

alphashape-package (alphashape), [1](#)

convex\_hull, [4](#), [7](#)

convex\_layer, [4](#)

delaunay, [5](#), [6](#)

find\_simplex, [6](#)

grid\_coordinates, [6](#)

in\_convex\_hull, [7](#)

voronoi, [8](#)