

# Package ‘alphahull’

January 2, 2012

**Title** Generalization of the convex hull of a sample of points in the plane

**Version** 0.2-1

**Date** 2009-10-21

**Author** Beatriz Pateiro-Lopez, Alberto Rodriguez-Casal.

**Maintainer** Beatriz Pateiro-Lopez <beatriz.pateiro@usc.es>

**Depends** tripack, sgeostat, splancs

**Suggests** deldir

**Description** This package computes the alpha-shape and alpha-convex hull of a given sample of points in the plane. The concepts of alpha-shape and alpha-convex hull generalize the definition of the convex hull of a finite set of points. The programming is based on the duality between the Voronoi diagram and Delaunay triangulation. The package also includes a function that returns the Delaunay mesh of a given sample of points and its dual Voronoi diagram in one single object.

**License** file LICENSE

**Repository** CRAN

**Date/Publication** 2011-10-23 10:08:35

## R topics documented:

alphahull-package . . . . .	2
ahull . . . . .	2
anglesArc . . . . .	5
arc . . . . .	5
areaahull . . . . .	6
ashape . . . . .	7
complement . . . . .	8
delvor . . . . .	10

dummycoor . . . . .	11
inahull . . . . .	12
inter . . . . .	13
koch . . . . .	14
lengthahull . . . . .	15
plot.ahull . . . . .	16
plot.ashape . . . . .	17
plot.delvor . . . . .	18
rkoch . . . . .	19
rotation . . . . .	20
trircircum . . . . .	21

<b>Index</b>	<b>22</b>
--------------	-----------

---

alphahull-package	<i>Generalization of the convex hull of a sample of points in the plane</i>
-------------------	---

---

## Description

This package computes the  $\alpha$ -shape and  $\alpha$ -convex hull of a given sample of points in the plane. The concepts of  $\alpha$ -shape and  $\alpha$ -convex hull generalize the definition of the convex hull of a finite set of points. The programming is based on the duality between the Voronoi diagram and Delaunay triangulation. The package also includes a function that returns the Delaunay mesh of a given sample of points and its dual Voronoi diagram in one single object.

## Details

Package: alphahull  
 Type: Package  
 Version: 0.2-1  
 Date: 2011-10-21  
 License: R functions: file LICENSE

## Author(s)

Beatriz Pateiro-Lopez, Alberto Rodriguez-Casal.  
 Maintainer: Beatriz Pateiro-Lopez <beatriz.pateiro@usc.es>

---

ahull	<i>alpha-convex hull calculation</i>
-------	--------------------------------------

---

**Description**

This function calculates the  $\alpha$ -convex hull of a given sample of points in the plane for  $\alpha > 0$ .

**Usage**

```
ahull(x, y = NULL, alpha)
```

**Arguments**

<code>x, y</code>	The <code>x</code> and <code>y</code> arguments provide the <code>x</code> and <code>y</code> coordinates of a set of points. Alternatively, a single argument <code>x</code> can be provided, see <a href="#">Details</a> .
<code>alpha</code>	Value of $\alpha$ .

**Details**

An attempt is made to interpret the arguments `x` and `y` in a way suitable for computing the  $\alpha$ -convex hull. Any reasonable way of defining the coordinates is acceptable, see [xy.coords](#).

The  $\alpha$ -convex hull is defined for any finite number of points. However, since the algorithm is based on the Delaunay triangulation, at least three non-collinear points are required.

If `y` is `NULL` and `x` is an object of class "delvor", then the  $\alpha$ -convex hull is computed with no need to invoke again the function [delvor](#) (it reduces the computational cost).

The complement of the  $\alpha$ -convex hull can be written as the union of  $O(n)$  open balls and halfplanes, see [complement](#). The boundary of the  $\alpha$ -convex hull is formed by arcs of open balls of radius  $\alpha$  (besides possible isolated sample points). The arcs are determined by the intersections of some of the balls that define the complement of the  $\alpha$ -convex hull. The extremes of an arc are given by  $c + rA_{\theta}v$  and  $c + rA_{-\theta}v$  where  $c$  and  $r$  represent the center and radius of the arc, respectively, and  $A_{\theta}v$  represents the clockwise rotation of angle  $\theta$  of the unitary vector  $v$ . Joining the end points of adjacent arcs we can define polygons that help us to determine the area of the estimator, see [areaahull](#).

**Value**

A list with the following components:

<code>arcs</code>	For each arc in the boundary of the $\alpha$ -convex hull, the columns of the matrix <code>arcs</code> store the center $c$ and radius $r$ of the arc, the unitary vector $v$ , the angle $\theta$ that define the arc and the indices of the end points, see <a href="#">Details</a> . The coordinates of the end points of the arcs are stored in <code>xahull</code> . For isolated points in the boundary of the $\alpha$ -convex hull, columns 3 to 6 of the matrix <code>arcs</code> are equal to zero.
<code>xahull</code>	A 2-column matrix with the coordinates of the original set of points besides possible new end points of the arcs in the boundary of the $\alpha$ -convex hull.
<code>length</code>	Length of the boundary of the $\alpha$ -convex hull, see <a href="#">lengthahull</a> .
<code>complement</code>	Output matrix from <a href="#">complement</a> .
<code>alpha</code>	Value of $\alpha$ .
<code>ashape.obj</code>	Object of class "ashape" returned by the function <a href="#">ashape</a> .

## References

- Edelsbrunner, H., Kirkpatrick, D.G. and Seidel, R. (1983). On the shape of a set of points in the plane. *IEEE Transactions on Information Theory*, 29(4), pp.551-559.
- Rodriguez-Casal, R. (2007). Set estimation under convexity type assumptions. *Annales de l'I.H.P.-Probabilites & Statistiques*, 43, pp.763-774.
- Pateiro-Lopez, B. (2008). *Set estimation under convexity type restrictions*. Phd. Thesis. Universidad de Santiago de Compostela. ISBN 978-84-9887-084-8.

## See Also

[plot.ahull](#).

## Examples

```
# Random sample in the unit square
x <- matrix(runif(100), nc = 2)
# Value of alpha
alpha <- 0.2
# Alpha-convex hull
ahull.obj <- ahull(x, alpha = alpha)
plot(ahull.obj)

# Uniform sample of size n=300 in the annulus B(c,0.5)\B(c,0.25),
# with c=(0.5,0.5).
n <- 300
theta<-runif(n,0,2*pi)
r<-sqrt(runif(n,0.25^2,0.5^2))
x<-cbind(0.5+r*cos(theta),0.5+r*sin(theta))
# Value of alpha
alpha <- 0.1
# Alpha-convex hull
ahull.obj <- ahull(x, alpha = alpha)
# The arcs defining the boundary of the alpha-convex hull are ordered
plot(x)
for (i in 1:dim(ahull.obj$arcs)[1]){
  arc(ahull.obj$arcs[i,1:2],ahull.obj$arcs[i,3],ahull.obj$arcs[i,4:5],
  ahull.obj$arcs[i,6],col=2)
  Sys.sleep(0.5)
}

# Random sample from a uniform distribution on a Koch snowflake
# with initial side length 1 and 3 iterations
x <- rkoch(2000, side = 1, niter = 3)
# Value of alpha
alpha <- 0.05
# Alpha-convex hull
ahull.obj <- ahull(x, alpha = alpha)
plot(ahull.obj)
```

---

anglesArc                      *Angles of the extremes of an arc*

---

**Description**

Given a vector  $v$  and an angle  $\theta$ , anglesArc returns the angles that  $A_\theta v$  and  $A_{-\theta} v$  form with the axis  $OX$ , where  $A_\theta v$  represents the clockwise rotation of angle  $\theta$  of the vector  $v$ .

**Usage**

```
anglesArc(v, theta)
```

**Arguments**

`v`                      Vector  $v$  in the plane.  
`theta`                  Angle  $\theta$  (in radians).

**Details**

The angle that forms the vector  $v$  with the axis  $OX$  takes its value in  $[0, 2\pi)$ .

**Value**

`angs`                  Numeric vector with two components.

**Examples**

```
# Let v=c(0,1) and theta=pi/4
# Consider the arc such that v is the internal angle bisector that
# divides the angle 2*theta into two equal angles
# The angles that the arc forms with the OX axis are pi/4 and 3*pi/4
v <- c(0,1)
theta <- pi/4
anglesArc(v, theta)
```

---

`arc`                      *Add an arc to a plot*

---

**Description**

This function adds the arc of  $B(c, r)$  between the angles that  $A_\theta v$  and  $A_{-\theta} v$  form with the axis  $OX$ , where  $A_\theta v$  represents the clockwise rotation of angle  $\theta$  of the vector  $v$ .

**Usage**

```
arc(c, r, v, theta, ...)
```

**Arguments**

<code>c</code>	Center $c$ of the arc.
<code>r</code>	Radius $r$ of the arc.
<code>v</code>	Vector $v$ in the plane.
<code>theta</code>	Angle $\theta$ (in radians).
<code>...</code>	Arguments to be passed to methods, such as graphical parameters (see <a href="#">par</a> ).

**See Also**

[plot.ahull](#).

**Examples**

```
# Plot of the circumference of radius 1
theta <- seq(0, 2*pi, length = 100)
r <- 1
plot(r*cos(theta), r*sin(theta), type = "l")
# Add in red the arc between pi/4 and 3*pi/4
arc(c(0,0), 1, c(0,1), pi/4, col = 2, lwd = 2)
```

---

areaahull

*Area of the alpha-convex hull*

---

**Description**

This function calculates the area of the  $\alpha$ -convex hull of a sample of points.

**Usage**

```
areaahull(x)
```

**Arguments**

<code>x</code>	Object of class "ahull".
----------------	--------------------------

**Value**

<code>area</code>	Area of the $\alpha$ -convex hull.
-------------------	------------------------------------

**See Also**

[ahull](#).

**Examples**

```
# Random sample in the unit square
x <- matrix(runif(500), nc = 2)
# Value of alpha
alpha <- 1
# alpha-convex hull
ahull.obj <- ahull(x, alpha = alpha)
# Area of the alpha-convex hull
areaahull(ahull.obj)
```

ashape

*alpha-shape hull calculation***Description**

This function calculates the  $\alpha$ -shape of a given sample for  $\alpha > 0$ .

**Usage**

```
ashape(x, y = NULL, alpha)
```

**Arguments**

<code>x, y</code>	The x and y coordinates of a set of points. Alternatively, a single argument <code>x</code> can be provided, see Details.
<code>alpha</code>	Value of $\alpha$ .

**Details**

An attempt is made to interpret the arguments `x` and `y` in a way suitable for computing the  $\alpha$ -shape, see [xy.coords](#).

The  $\alpha$ -shape is defined for any finite number of points. However, since the algorithm is based on the Delaunay triangulation, at least three non-collinear points are required.

If `y` is `NULL` and `x` is an object of class "delvor", then the  $\alpha$ -shape is computed without invoking again the function `delvor` (it reduces the computational cost).

The function `ashape` returns (among other values) the matrix `edges`. The structure of `edges` is that of matrix `mesh` returned by the function `delvor`. Note that the  $\alpha$ -shape is a subgraph of the Delaunay triangulation and, therefore, `edges` is a submatrix of `mesh`.

**Value**

A list with the following components:

<code>edges</code>	A <i>n.seg</i> -row matrix with the coordinates and indexes of the edges of the Delaunay triangulation that form the $\alpha$ -shape. The number of rows <i>n.seg</i> coincides with the number of segments of the $\alpha$ -shape. The matrix also includes information of the Voronoi extremes corresponding to each segment.
--------------------	---

length	Length of the $\alpha$ -shape.
alpha	Value of $\alpha$ .
alpha.extremes	Vector with the indexes of the sample points that are $\alpha$ -extremes. See Edelsbrunner <i>et al.</i> (1983).
delvor.obj	Object of class "delvor" returned by the <a href="#">delvor</a> function.
x	A 2-column matrix with the coordinates of the set of points.

## References

Edelsbrunner, H., Kirkpatrick, D.G. and Seidel, R. (1983). On the shape of a set of points in the plane. *IEEE Transactions on Information Theory*, 29(4), pp.551-559.

## See Also

[plot.ashape](#), [delvor](#).

## Examples

```
# Uniform sample of size n=300 in the annulus B(c,0.5)\B(c,0.25),
# with c=(0.5,0.5).
n <- 300
theta<-runif(n,0,2*pi)
r<-sqrt(runif(n,0.25^2,0.5^2))
x<-cbind(0.5+r*cos(theta),0.5+r*sin(theta))
# Value of alpha
alpha <- 0.1
# alpha-shape
ashape.obj <- ashape(x, alpha = alpha)
# If we change the value of alpha there is no need to compute
# again the Delaunay triangulation and Voronoi Diagram
alpha <- 0.4
ashape.obj.new <- ashape(ashape.obj$delvor.obj, alpha = alpha)

# Random sample from a uniform distribution on a Koch snowflake
# with initial side length 1 and 3 iterations
x <- rkoch(2000, side = 1, niter = 3)
# Value of alpha
alpha <- 0.05
# alpha-shape
ashape.obj <- ashape(x, alpha = alpha)
```

---

complement

*Complement of the alpha-convex hull*

---

## Description

This function calculates the complement of the  $\alpha$ -convex hull of a given sample for  $\alpha > 0$ .

**Usage**

```
complement(x, y = NULL, alpha)
```

**Arguments**

<code>x</code> , <code>y</code>	The <code>x</code> and <code>y</code> arguments provide the <code>x</code> and <code>y</code> coordinates of a set of points. Alternatively, a single argument <code>x</code> can be provided, see Details.
<code>alpha</code>	Value of $\alpha$ .

**Details**

An attempt is made to interpret the arguments `x` and `y` in a way suitable for computing the  $\alpha$ -shape. Any reasonable way of defining the coordinates is acceptable, see [xy.coords](#).

If `y` is `NULL` and `x` is an object of class "delvor", then the complement of the  $\alpha$ -convex hull is computed with no need to invoke again the function [delvor](#) (it reduces the computational cost).

The complement of the  $\alpha$ -convex hull is calculated as a union of open balls and halfplanes that do not contain any point of the sample. See Edelsbrunner *et al.* (1983) for a basic description of the algorithm. The construction of the complement is based on the Delaunay triangulation and Voronoi diagram of the sample, provided by the function [delvor](#). The function [complement](#) returns a matrix `compl`. For each row `i`, `compl[i, ]` contains the information relative to an open ball or halfplane of the complement. The first three columns are assigned to the characterization of the ball or halfplane `i`. The information relative to the edge of the Delaunay triangulation that generates the ball or halfplane `i` is contained in `compl[i, 4:16]`. Thus, if the row `i` refers to an open ball, `compl[i, 1:3]` contains the center and radius of the ball. Furthermore, `compl[i, 17:18]` and `compl[i, 19]` refer to the unitary vector  $v$  and the angle  $\theta$  that characterize the arc that joins the two sample points that define the ball `i`. If the row `i` refers to a halfplane, `compl[i, 1:3]` determines its equation. For the halfplane  $y > a + bx$ , `compl[i, 1:3]=(a,b,-1)`. In the same way, for the halfplane  $y < a + bx$ , `compl[i, 1:3]=(a,b,-2)`, for the halfplane  $x > a$ , `compl[i, 1:3]=(a,0,-3)` and for the halfplane  $x < a$ , `compl[i, 1:3]=(a,0,-4)`.

**Value**

<code>compl</code>	Output matrix. For each row <code>i</code> , <code>compl[i, ]</code> contains the information relative to an open ball or halfplane of the complement of the $\alpha$ -convex hull, see Details.
--------------------	--

**References**

Edelsbrunner, H., Kirkpatrick, D.G. and Seidel, R. (1983) *On the shape of a set of points in the plane*. IEEE Transactions on Information Theory, Vol IT-29, No. 4.

**See Also**

[delvor](#), [ahull](#).

## Examples

```
# Random sample in the unit square
x <- matrix(runif(100), nc = 2)
# Value of alpha
alpha <- 0.2
# Complement of the alpha-convex hull
compl <- complement(x, alpha = alpha)
```

---

delvor

*Delaunay triangulation and Voronoi diagram*

---

## Description

This function returns a matrix with information about the Delaunay triangulation and Voronoi diagram of a given sample.

## Usage

```
delvor(x, y = NULL)
```

## Arguments

`x`, `y`            The `x` and `y` arguments provide the `x` and `y` coordinates of a set of points. Alternatively, a single argument `x` can be provided, see Details.

## Details

An attempt is made to interpret the arguments `x` and `y` in a way suitable for computing the Delaunay triangulation and Voronoi diagram. Any reasonable way of defining the coordinates is acceptable, see [xy.coords](#).

The function `tri.mesh` from package **tripack** calculates the Delaunay triangulation of at least three non-collinear points using Fortran functions from the library TRIPACK. Using the Delaunay triangulation, the function `delvor` calculates the corresponding Voronoi diagram. For each edge of the Delaunay triangulation there is a segment in the Voronoi diagram, given by the union of the circumcenters of the two neighbour triangles that share the edge. For those triangles with edges on the convex hull, the corresponding line in the Voronoi diagram is a semi-infinite segment, whose boundless extreme is calculated by the function `dummycoor`. The function `delvor` returns the sample, the output object of class "tri" from the function `tri.mesh` and a matrix `mesh` with all the necessary information of the Delaunay triangulation and Voronoi diagram. Thus, for each edge of the Delaunay triangulation the output matrix contains the indexes and coordinates of the sample points that form the edge, the indexes and coordinates of the extremes of the corresponding segment in the Voronoi diagram, and an indicator that takes the value 1 for those extremes of the Voronoi diagram that represent a boundless extreme.

**Value**

A list with the following components:

mesh	A $n.edges$ -row matrix, where $n.edges$ is the total number of different edges of the Delaunay triangulation.
x	A 2-column matrix with the coordinates of the sample points.
tri.obj	Object of class "tri". See <code>tri.mesh</code> in package <b>tripack</b> .

**References**

Renka, R. J. (1996). Algorithm 751: TRIPACK: a constrained two-dimensional Delaunay triangulation package, *ACM Trans. Math. Softw.*, 22(1), pp.1-8.

**See Also**

[plot.delvor](#).

**Examples**

```
# Simple example from TRIPACK
data(tritest)
# Delaunay triangulation and Voronoi diagram calculation
delvor.obj <- delvor(tritest)

# Random sample in the unit square
x <- matrix(runif(20), nc = 2)
# Delaunay triangulation and Voronoi diagram calculation
delvor.obj <- delvor(x)
```

---

dummycoor

*Semi-infinite edge of the Voronoi diagram*

---

**Description**

This function determines fictitious coordinates for the boundless extreme of a semi-infinite edge of the Voronoi diagram.

**Usage**

```
dummycoor(tri.obj, l1, l2, m, away)
```

**Arguments**

tri.obj	Object of class "tri". See <a href="#">tri.mesh</a> in package <b>tripack</b> .
l1	Index of the sample point corresponding to one vertex of a triangle of Delaunay that lies on the convex hull, see Details.
l2	Index of the sample point corresponding to other vertex of a triangle of Delaunay that lies on the convex hull, see Details.
m	Index of the circumcenter of the triangle of Delaunay with one edge on the convex hull.
away	Constant that determines how far away the fictitious boundless extreme is located.

**Details**

When a triangle of the Delaunay triangulation has one of its edges (given by the segment that joins the sample points with indexes l1 and l2) on the convex hull, the corresponding segment of the Voronoi diagram is semi-infinite. The finite extreme coincides with the circumcenter of the triangle and the direction of the line is given by the perpendicular bisector of the edge that lies on the convex hull.

**Value**

dum	Fictitious coordinates of the boundless extreme.
-----	--

**See Also**

[delvor](#).

---

 inahull

*Determine whether a point belongs to the alpha-convex hull*

---

**Description**

This function determines whether a given point  $p$  belongs to the  $\alpha$ -convex hull of a sample.

**Usage**

```
inahull(ahull.obj, p)
```

**Arguments**

ahull.obj	Object of class "ahull" returned by the function <a href="#">ahull</a> .
p	Numeric vector with two components describing a point in the plane.

**Details**

The complement of the  $\alpha$ -convex hull of a sample is calculated by [complement](#). The function [inahull](#) checks whether the point  $p$  belongs to any of the open balls or halfplanes that define the complement.

**Value**

`in.ahull` A logical value specifying whether the point  $p$  belongs to the  $\alpha$ -convex hull.

**See Also**

[ahull](#), [complement](#).

**Examples**

```
# Random sample in the unit square
x <- matrix(runif(100), nc = 2)
# Value of alpha
alpha <- 0.2
# alpha-convex hull
ahull.obj <- ahull(x, alpha = alpha)
# Check if the point (0.5, 0.5) belongs to the alpha-convex hull
inahull(ahull.obj, p = c(0.5, 0.5))
```

---

inter

*Intersection of two circumferences*


---

**Description**

This function calculates the intersection of two circumferences, given their centers and radius  $c1, r1$  and  $c2, r2$ , respectively.

**Usage**

```
inter(c11, c12, r1, c21, c22, r2)
```

**Arguments**

<code>c11</code>	X-coordinate of the center $c1$ .
<code>c12</code>	Y-coordinate of the center $c1$ .
<code>r1</code>	Radius $r1$ .
<code>c21</code>	X-coordinate of the center $c2$ .
<code>c22</code>	Y-coordinate of the center $c2$ .
<code>r2</code>	Radius $r2$ .

**Details**

The function `inter` is internally called by the function [ahull](#).

**Value**

A list with the following components:

<code>n.cut</code>	Number of intersection points (0,1,2, or Inf).
<code>v1</code>	If there are two intersection points, <code>v1</code> is the numeric vector whose components are the coordinates of the unitary vector that has its origin in <code>c1</code> and it's perpendicular to the chord that joins the intersection points of the two circumferences. Otherwise, <code>v1=(0,0)</code>
<code>theta1</code>	Angle that forms <code>v1</code> with the radius that joins the center <code>c1</code> with an intersection point.
<code>v2</code>	If there are two intersection points, <code>v2</code> is the numeric vector whose components are the coordinates of the unitary vector that has its origin in <code>c2</code> and it's perpendicular to the chord that joins the intersection points of the two circumferences. Otherwise, <code>v2=(0,0)</code>
<code>theta2</code>	Angle that forms <code>v2</code> with the radius that joins the center <code>c2</code> with an intersection point.

---

<code>koch</code>	<i>Construct a Koch snowflake curve</i>
-------------------	---

---

**Description**

This function uses recursion to construct a Koch snowflake curve.

**Usage**

```
koch(side = 3, niter = 5)
```

**Arguments**

<code>side</code>	Side length of the initial equilateral triangle.
<code>niter</code>	Number of iterations in the development of the snowflake curve.

**Details**

The Koch snowflake is a fractal curve described by the Swedish mathematician Helge von Koch in 1904. It is built by starting with an equilateral triangle, removing the inner third of each side, building another equilateral triangle at the location where the side was removed, and then repeating the process.

**Value**

`vertices` A 2-column matrix with the coordinates of the snowflake vertices.

## References

von Koch, H. (1904). Sur une courbe continue sans tangente, obtenue par une construction geometrique elementaire. *Arkiv for Matematik*, 1, pp.681-704.

## See Also

[rkoch](#).

## Examples

```
# The first four iterations of a Koch snowflake
# with side length of the initial equilateral triangle equal to 3.
vertices <- koch(side = 2, niter = 4)
plot(vertices[, 1], vertices[, 2], type = "l", asp = TRUE,
      main = "Koch snowflake", xlab = "", ylab = "", col = 4)
polygon(vertices[, 1], vertices[, 2], col = 4)
```

---

lengthahull

*Length of the boundary of the alpha-convex hull*

---

## Description

This function calculates the length of the boundary of the  $\alpha$ -convex hull of a given sample.

## Usage

```
lengthahull(ahull.arcs)
```

## Arguments

ahull.arcs      Output matrix of arcs returned by [ahull](#).

## Details

The function lengthahull is internally called by the function [ahull](#).

## Value

length          Length of the boundary of the  $\alpha$ -convex hull.

## See Also

[ahull](#).

**Examples**

```
# Random sample in the unit square
x <- matrix(runif(100), nc = 2)
# Value of alpha
alpha <- 0.2
# alpha-convex hull
ahull.obj <- ahull(x, alpha = alpha)
# Length of the alpha-convex hull
ahull.obj$length
```

---

plot.ahull

*Plot the alpha-convex hull*


---

**Description**

This function returns a plot of the  $\alpha$ -convex hull. If desired, it also adds the Delaunay triangulation, Voronoi diagram and  $\alpha$ -shape of the sample.

**Usage**

```
## S3 method for class 'ahull'
plot(x, add = FALSE, do.shape = FALSE,
      wlines = c("none", "both", "del", "vor"), wpoints = TRUE,
      number = FALSE, col = NULL, xlim = NULL,
      ylim = NULL, lwd = NULL, ...)
```

**Arguments**

x	Object of class "ahull".
add	Logical, if TRUE add to a current plot.
do.shape	Logical, indicates if the $\alpha$ -shape should also be plotted.
wlines	"Which lines?". I.e. should the Delaunay triangulation be plotted (wlines='del'), should the Voronoi diagram be plotted (wlines='vor'), should both be plotted (wlines='both'), or none (wlines='none', the default)?
wpoints	Logical, indicates if sample points should be plotted.
number	Logical, defaulting to FALSE; if TRUE then the points plotted will be labelled with their index numbers.
col	The colour numbers for plotting the $\alpha$ -convex hull, $\alpha$ -shape, data points, Delaunay triangulation, Voronoi diagram, and the point numbers, in that order; defaults to c(1,1,1,1,1,1). If fewer than six numbers are given, they are recycled. (If more than six numbers are given, the redundant ones are ignored.)
xlim	The limits on the x-axis.
ylim	The limits on the y-axis.
lwd	The line widths for plotting the tessellations, the $\alpha$ -shape, and the $\alpha$ -convex hull, in that order; defaults to c(1,1,2).
...	Arguments to be passed to methods, such as graphical parameters (see <a href="#">par</a> ).

**See Also**

[ahull](#), [ashape](#).

**Examples**

```
# Random sample in the unit square
x <- matrix(runif(100), nc = 2)
# Value of alpha
alpha <- 0.2
# alpha-convex hull
ahull.obj <- ahull(x, alpha = alpha)
# Plot including the alpha-convex hull in pink, alpha-shape in blue,
# sample points in black, voronoi diagram in green
# and Delaunay triangulation in red
plot(ahull.obj, do.shape = TRUE, wlines = "both", col = c(6, 4, 1, 2, 3))

# Random sample from a uniform distribution on a Koch snowflake
# with initial side length 1 and 3 iterations
x <- rkoch(2000, side = 1, niter = 3)
# Value of alpha
alpha <- 0.05
# Alpha-convex hull
ahull.obj <- ahull(x, alpha = alpha)
plot(ahull.obj)
```

---

plot.ashape

*Plot the alpha-shape*

---

**Description**

This function returns a plot of the  $\alpha$ -shape. If desired, it also adds the Delaunay triangulation and Voronoi diagram of the sample.

**Usage**

```
## S3 method for class 'ashape'
plot(x, add = FALSE, wlines = c("none", "both", "del", "vor"),
     wpoints = TRUE, number = FALSE, col = NULL,
     xlim = NULL, ylim = NULL, lwd = NULL, ...)
```

**Arguments**

x	Object of class "ashape".
add	Logical, if TRUE add to a current plot.
wlines	"Which lines?". I.e. should the Delaunay triangulation be plotted (wlines='del'), should the Voronoi diagram be plotted (wlines='vor'), should both be plotted (wlines='both'), or none (wlines='none', the default)?

wpoints	Logical, indicates if sample points should be plotted.
number	Logical, defaulting to FALSE; if TRUE then the points plotted will be labelled with their index numbers.
col	The colour numbers for plotting the $\alpha$ -shape, data points, Delaunay triangulation, Voronoi diagram, and the point numbers, in that order; defaults to c(1,1,1,1,1). If fewer than five numbers are given, they are recycled. (If more than five numbers are given, the redundant ones are ignored.)
xlim	The limits on the x-axis.
ylim	The limits on the y-axis.
lwd	The line widths for plotting the tessellations and the $\alpha$ -shape; defaults to c(1,2).
...	Arguments to be passed to methods, such as graphical parameters (see <a href="#">par</a> ).

**See Also**

[ashape](#).

**Examples**

```
# Uniform sample of size n=300 in the annulus B(c, 0.5)\B(c, 0.25)
# with c=(0.5, 0.5).
n <- 300
theta<-runif(n,0,2*pi)
r<-sqrt(runif(n,0.25^2,0.5^2))
x<-cbind(0.5+r*cos(theta),0.5+r*sin(theta))
# Value of alpha
alpha <- 0.1
# alpha-shape
ashape.obj <- ashape(x, alpha = alpha)
# Plot alpha-shape in blue, sample points in black,
# and Delaunay triangulation in red
plot(ashape.obj, wlines= "del", col = c(4, 1, 2))

# Random sample from a uniform distribution on a Koch snowflake
# with initial side length 1 and 3 iterations
x <- rkoch(2000, side = 1, niter = 3)
# Value of alpha
alpha <- 0.05
# alpha-shape
ashape.obj <- ashape(x, alpha = alpha)
# Plot alpha-shape in blue
plot(ashape.obj, col = c(4, 1))
```

---

plot.delvor

*Plot the Voronoi diagram and Delaunay traingulation*

---

**Description**

This function returns a plot of the Voronoi diagram and Delaunay traingulation.

**Usage**

```
## S3 method for class 'delvor'
plot(x, add = FALSE, wlines = c("both", "del", "vor"),
     wpoints = TRUE, number = FALSE, col = NULL,
     xlim = NULL, ylim = NULL, ...)
```

**Arguments**

<code>x</code>	An object of class "delvor" as constructed by the function <code>delvor</code> .
<code>add</code>	Logical, if TRUE add to a current plot.
<code>wlines</code>	"Which lines?". I.e. should the Delaunay triangulation be plotted ( <code>wlines='del'</code> ), should the Voronoi diagram be plotted ( <code>wlines='vor'</code> ), or should both be plotted ( <code>wlines='both'</code> , the default)?
<code>wpoints</code>	Logical, indicates if sample points should be plotted.
<code>number</code>	Logical, defaulting to FALSE; if TRUE then the points plotted will be labelled with their index numbers.
<code>col</code>	The colour numbers for plotting the data points, Delaunay triangulation, Voronoi diagram, and the point numbers, in that order; defaults to <code>c(1,1,1,1)</code> . If fewer than four numbers are given, they are recycled. (If more than four numbers are given, the redundant ones are ignored.)
<code>xlim</code>	The limits on the x-axis.
<code>ylim</code>	The limits on the y-axis.
<code>...</code>	Arguments to be passed to methods, such as graphical parameters (see <a href="#">par</a> ).

**See Also**

[delvor](#).

**Examples**

```
# Random sample in the unit square
x <- matrix(runif(100), nc = 2)
# Delaunay triangulation and Voronoi diagram
delvor.obj <- delvor(x)
# Plot Voronoi diagram and Delaunay triangulation
plot(delvor.obj)
```

---

 rkoch

*Random generation on a Koch snowflake curve*


---

**Description**

This function generates random points from a uniform distribution on a Koch snowflake.

**Usage**

```
rkoch(n, side = 3, niter = 5)
```

**Arguments**

n	Number of observations.
side	Side length of the initial equilateral triangle of the Koch snowflake curve.
niter	Number of iterations in the development of the snowflake curve.

**Value**

A 2-column matrix with the coordinates of generated points.

**See Also**

[koch](#).

**Examples**

```
unifkoch <- rkoch(2000, side = 1, niter = 3)
plot(unifkoch, asp = TRUE)
```

---

rotation

*Clockwise rotation*

---

**Description**

This function calculates the clockwise rotation of angle  $\theta$  of a given vector  $v$  in the plane.

**Usage**

```
rotation(v, theta)
```

**Arguments**

v	Vector $v$ in the plane.
theta	Angle $\theta$ (in radians).

**Value**

v.rot            Vector after rotation.

**Examples**

```
# Rotation of angle pi/4 of the vector (0,1)
rotation(v = c(0, 1), theta = pi/4)
```

---

trircircum	Returns the circumcentres of a Delaunay triangulation
------------	---

---

### Description

This function converts a triangulation data structure to a triangle list with vertexes, opposite triangles and arcs indexes. It also returns the coordinates of the circumcentre of each triangle.

### Usage

```
trircircum(tri.obj)
```

### Arguments

`tri.obj` Object of class "tri", see [tri.mesh](#) from package **tripack**.

### Details

The function [trircircum](#) calls the Fortran function `delaunaycircum` and it is internally used in the function [delvor](#).

### Value

`tri.info` A *n.tri*-row matrix, where *n.tri* is the total number of different triangles of the Delaunay triangulation. For each row *i*, `tri.info[i,]` contains the vertex nodal indexes (first three columns), the indexes of the triangles and arcs which are opposite to the triangle *i* (columns 4-9) and the coordinates of the circumcentre of the triangle *i* (columns 10-11).

### References

Renka, R. J. (1996). Algorithm 751: TRIPACK: a constrained two-dimensional Delaunay triangulation package, *ACM Trans. Math. Softw.*, 22(1), pp.1-8.

### Examples

```
# Random sample in the unit square
x <- runif(20)
y <- runif(20)
del <- trircircum(tri.mesh(x, y))
```

# Index

## \*Topic **nonparametric**

- ahull, [2](#)
  - alphahull-package, [2](#)
  - anglesArc, [5](#)
  - arc, [5](#)
  - areaahull, [6](#)
  - ashape, [7](#)
  - complement, [8](#)
  - delvor, [10](#)
  - dummycoor, [11](#)
  - inahull, [12](#)
  - inter, [13](#)
  - koch, [14](#)
  - lengthahull, [15](#)
  - plot.ahull, [16](#)
  - plot.ashape, [17](#)
  - plot.delvor, [18](#)
  - rkoch, [19](#)
  - rotation, [20](#)
  - tricircum, [21](#)
- ahull, [2](#), [6](#), [9](#), [12–15](#), [17](#)
- alphahull-package, [2](#)
- anglesArc, [5](#)
- arc, [5](#)
- areaahull, [3](#), [6](#)
- ashape, [3](#), [7](#), [7](#), [17](#), [18](#)
- complement, [3](#), [8](#), [9](#), [13](#)
- delvor, [3](#), [7–9](#), [10](#), [12](#), [19](#), [21](#)
- dummycoor, [10](#), [11](#)
- inahull, [12](#), [13](#)
- inter, [13](#)
- koch, [14](#), [20](#)
- lengthahull, [3](#), [15](#)
- par, [6](#), [16](#), [18](#), [19](#)
- plot.ahull, [4](#), [6](#), [16](#)
- plot.ashape, [8](#), [17](#)
- plot.delvor, [11](#), [18](#)
- rkoch, [15](#), [19](#)
- rotation, [20](#)
- tri.mesh, [10–12](#), [21](#)
- tricircum, [21](#), [21](#)
- xy.coords, [3](#), [7](#), [9](#), [10](#)