

Package ‘compGeometeR’

May 25, 2020

Type Package

Title Create Computational Geometry Algorithms for use in R

Version 1.0

Date 2020-05-15

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

Description Create Computational Geometry Algorithms for use in R

License GPU

Depends R (>= 3.5.1)

Imports devtools

Encoding UTF-8

LazyData true

RoxygenNote 7.1.0

Suggests testthat

NeedsCompilation no

R topics documented:

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

Index	11
--------------	-----------

alpha_complex	<i>Alpha complex</i>
---------------	----------------------

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 and the alpha complex is equivalent to a Delaunay triangulation.

Value

Returns a list consisting of:

- input_points: the input points used to create the Voronoi diagram.
- simplices: a s -by- $d + 1$ matrix of point indices that define the s **simplices** that make up the alpha complex.
- circumcentres: a s -by- d matrix of coordinates that define the centre of the **circumcircle** associated with each simplex.
- circumradii: the radius of each circumcircle.

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")
```

compGeometreR

Computational geometry algorithms for R

Description

Implementation of various computation geomtery algorithms for use in R

Package:	compGeometreR
Date:	2020-05-20
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>

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:

- input_points: the input points used to create the convex hull.

- `hull_simplices`: a s -by- d matrix of point indices that define the s **simplices** that make up the convex hull.
- `hull_indices`: a vector of the point indices that form the convex hull.
- `hull_vertices`: a matrix of point coordinates that form 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 (s in seq(nrow(ch$hull_simplices))) {
  lines(ch$input_points[ch$hull_simplices[s, ], ], 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

- | | |
|---------------------|---|
| <code>points</code> | a n -by- d dataframe or matrix. The rows represent n points and the d columns the coordinates in d -dimensional space. |
| <code>layer</code> | 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:

- `input_points`: the input points used to create the convex layer.
- `hull_simplices`: a s -by- d matrix of point indices that define the s **simplices** that make up the convex layer.
- `hull_indices`: a vector of the point indices that form the convex layer.
- `hull_vertices`: a matrix of point coordinates that form 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 (s in seq(nrow(cl$hull_simplices))) {
    lines(cl$input_points[cl$hull_simplices[s, ], ], 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:

- `input_points`: the input points used to create the Delaunay triangulation .
- `simplices`: a s -by- $d + 1$ matrix of point indices that define the s **simplices** that make up the Delaunay triangulation.
- `simplex_neighs`: a list containing for each simplex the neighbouring simplices.

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

Usage

```
find_simplex(simplices, test_points)
```

Arguments

<code>simplices</code>	A Delaunay trigulation list object created by <code>delaunay</code> or a alpha complex list object created by <code>alpha_complex</code> that contain simplices.
<code>test_points</code>	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 simplices.

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(simplices = 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", "simplices", "test points"),
      text.col=c("black", "red", "blue"), title = "Indices 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	<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

<code>hull</code>	A convex hull list object created by <code>convex_hull</code>
<code>test_points</code>	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. If any of the test point coordinates contain NA then the output is also NA.

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)
plot(p, pch = as.character(seq(nrow(p))), xlim=c(0,100), ylim=c(0,100))
for (s in seq(nrow(ch$hull_simplices))) {
  lines(ch$input_points[ch$hull_simplices[s, ], ], col = "red")
}
# Check if some test points are in the convex hull
p_test <- data.frame(c(20, 50, 60, 90), c(20, 60, 60, 40))
points(p_test[,1], p_test[,2], pch=c("1", "2", "3", "4"), col="blue")
legend("topright", legend = c("input points", "test points"),
      text.col=c("black", "blue"), title = "Indices for:", bty="n")
p_test_hull <- in_convex_hull(hull = ch, test_points = p_test)
print(p_test_hull)
```

 voronoi

 Voronoi diagram

Description

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

Usage

```
voronoi(points = NULL, delaunay = FALSE)
```

Arguments

- | | |
|----------|--|
| points | a n -by- d dataframe or matrix. The rows represent n points and the d columns the coordinates in d -dimensional space. |
| delaunay | a boolean indicating if the Delaunay triangulation, which is the dual of the Voronoi diagram should also be returned, defaults to FALSE. |

Value

Returns a list consisting of:

- `input_points`: the input points used to create the Voronoi diagram.
- `voronoi_vertices`: a i -by- d matrix of point coordinates that define the vertices that make each Voronoi region v .
- `voronoi_regions`: a list of length p that for each input point contains indices for the Voronoi vertices that define the Voronoi region v for each input point - if the indices include zeros then the Voronoi region is infinite.

Additionally, if `delaunay = TRUE` the returned list also includes:

- `simplices`: a s -by- $d + 1$ matrix of point indices that define the s **simplices** that make up the Delaunay triangulation.
- `circumradii`: for each simplex the radius of the associated **circumcircle** (note: the `voronoi_vertices` are equivalent to the centres of the circumcircles).
- `simplex_neighs`: a list containing for each simplex the neighbouring simplices.

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

[delaunay](#)

Examples

```
# Define points
x <- c(30, 70, 20, 50, 40, 70, 20, 55, 30)
y <- c(35, 80, 70, 50, 60, 20, 20, 55, 65)
p <- data.frame(x, y)
# Create Voronoi diagram and plot
vd <- voronoi(points = p)
cols = c("red", "blue", "green", "darkgrey", "purple", "lightseagreen",
        "brown", "darkgreen", "orange")
plot(vd$input_points, pch = as.character(seq(nrow(p))), col=cols,
     xlim=c(0,100), ylim=c(0,100))
text(vd$voronoi_vertices[,1], vd$voronoi_vertices[,2],
     labels = as.character(seq(nrow(vd$voronoi_vertices))))
r = 0
for (vd_region in vd$voronoi_regions) {
  r = r + 1
  if (!0 %in% vd_region) {
    polygon(vd$voronoi_vertices[vd_region,], density=20, col = cols[r])
  }
}
```

Index

*Topic **package**
 compGeometreR, 3

alpha_complex, 2, 6

compGeometreR, 3
convex_hull, 3, 5, 8
convex_layer, 4, 4

delaunay, 5, 6, 9

find_simplex, 6

grid_coordinates, 7

in_convex_hull, 8

voronoi, 9