Package ‘geometry’

March 27, 2019

License GPL (>= 3)

Title Mesh Generation and Surface Tessellation

Description Makes the 'Qhull' library <http://www.qhull.org>
available in R, in a similar manner as in Octave and MATLAB. Qhull
computes convex hulls, Delaunay triangulations, halfspace
intersections about a point, Voronoi diagrams, furthest-site
Delaunay triangulations, and furthest-site Voronoi diagrams. It
runs in 2D, 3D, 4D, and higher dimensions. It implements the
Quickhull algorithm for computing the convex hull. Qhull does not
support constrained Delaunay triangulations, or mesh generation of
non-convex objects, but the package does include some R functions
that allow for this.

Version 0.4.1

URL https://davidcsterratt.github.io/geometry

Date 2019-03-27

BugReports https://github.com/davidcsterratt/geometry/issues

Depends R (>= 3.0.0)

Imports magic, Rcpp, lpSolve

Suggests spelling, testthat, rgl, R.matlab, tripack

RoxygenNote 6.1.1

LinkingTo Rcpp, RcppProgress

Encoding UTF-8

Language en-GB

NeedsCompilation yes

Author Jean-Romain Roussel [cph, ctb] (wrote tsearch function with QuadTrees),
C. B. Barber [cph],
Kai Habel [cph, aut],
Raoul Grasman [cph, aut],
Robert B. Gramacy [cph, aut],
Pavlo Mozharovskyi [cph, aut],
David C. Sterratt [cph, aut, cre]
(<https://orcid.org/0000-0001-9092-9099>)

Maintainer David C. Sterratt <david.c.sterratt@ed.ac.uk>

Repository CRAN

Date/Publication 2019-03-27 17:10:03 UTC

R topics documented:

`bary2cart` *Conversion of Barycentric to Cartesian coordinates*

Description

Given the barycentric coordinates of one or more points with respect to a simplex, compute the Cartesian coordinates of these points.

Usage

```
bary2cart(X, Beta)
```

Arguments

`X`	Reference simplex in N dimensions represented by a $N+1$-by-N matrix
`Beta`	M points in barycentric coordinates with respect to the simplex `X` represented by a M-by-$N+1$ matrix

Value

M-by-N matrix in which each row is the Cartesian coordinates of corresponding row of `Beta`

Author(s)

David Sterratt

See Also

`cart2bary`

Examples

```
## Define simplex in 2D (i.e. a triangle)
X <- rbind(c(0, 0),
           c(0, 1),
           c(1, 0))
## Cartesian cooridinates of points
beta <- rbind(c(0, 0.5, 0.5),
              c(0.1, 0.8, 0.1))
## Plot triangle and points
trimesh(rbind(1:3), X)
text(X[,1], X[,2], 1:3) # Label vertices
P <- bary2cart(X, beta)
points(P)
```

cart2bary *Conversion of Cartesian to Barycentric coordinates.*

Description

Given the Cartesian coordinates of one or more points, compute the barycentric coordinates of these points with respect to a simplex.

Usage

```
cart2bary(X, P)
```

Arguments

X	Reference simplex in N dimensions represented by a $N+1$-by-N matrix
P	M-by-N matrix in which each row is the Cartesian coordinates of a point.

Details

Given a reference simplex in N dimensions represented by a $N+1$-by-N matrix an arbitrary point P in Cartesian coordinates, represented by a 1-by-N row vector, can be written as

$$P = \beta X$$

where β is an $N+1$ vector of the barycentric coordinates. A criterion on β is that

$$\sum_i \beta_i = 1$$

Now partition the simplex into its first N rows X_N and its $N+1$th row X_{N+1}. Partition the barycentric coordinates into the first N columns β_N and the $N+1$th column β_{N+1}. This allows us to write

$$P_{N+1} - X_{N+1} = \beta_N X_N + \beta_{N+1} X_{N+1} - X_{N+1}$$

which can be written

$$P_{N+1} - X_{N+1} = \beta_N (X_N - 1_N X_{N+1})$$

where 1_N is an N-by-1 matrix of ones. We can then solve for β_N:

$$\beta_N = (P_{N+1} - X_{N+1})(X_N - 1_N X_{N+1})^{-1}$$

and compute

$$\beta_{N+1} = 1 - \sum_{i=1}^{N} \beta_i$$

This can be generalised for multiple values of P, one per row.

Value

M-by-$N+1$ matrix in which each row is the barycentric coordinates of corresponding row of `P`. If the simplex is degenerate a warning is issued and the function returns `NULL`.

Note

Based on the Octave function by David Bateman.

Author(s)

David Sterratt

See Also

`bary2cart`

Examples

```
## Define simplex in 2D (i.e. a triangle)
X <- rbind(c(0, 0),
           c(0, 1),
           c(1, 0))
## Cartesian cooridinates of points
P <- rbind(c(0.5, 0.5),
           c(0.1, 0.8))
## Plot triangle and points
trimesh(rbind(1:3), X)
text(X[,1], X[,2], 1:3) # Label vertices
points(P)
cart2bary(X, P)
```

`cart2pol` *Transform Cartesian coordinates to polar or cylindrical coordinates.*

Description

The inputs `x`, `y` (, and `z`) must be the same shape, or scalar. If called with a single matrix argument then each row of `C` represents the Cartesian coordinate (`x`, `y` (, `z`)).

Usage

```
cart2pol(x, y = NULL, z = NULL)
```

Arguments

`x`	x-coordinates or matrix with three columns
`y`	y-coordinates (optional, if `x`) is a matrix
`z`	z-coordinates (optional, if `x`) is a matrix

Value

A matrix `P` where each row represents one polar/(cylindrical) coordinate (`theta`, `r`, (, `z`)).

Author(s)

Kai Habel

David Sterratt

See Also

`pol2cart`, `cart2sph`, `sph2cart`

---

cart2sph — *If called with a single matrix argument then each row of* `c` *represents the Cartesian coordinate (*`x`*,* `y`*,* `z`*).*

Description

Transform Cartesian to spherical coordinates

Usage

```
cart2sph(x, y = NULL, z = NULL)
```

Arguments

`x`	x-coordinates or matrix with three columns
`y`	y-coordinates (optional, if `x`) is a matrix
`z`	z-coordinates (optional, if `x`) is a matrix

Value

Matrix with columns:

`theta`	the angle relative to the positive x-axis
`phi`	the angle relative to the xy-plane
`r`	the distance to the origin `(0, 0, 0)`

Author(s)

Kai Habel

David Sterratt

See Also

`sph2cart`, `cart2pol`, `pol2cart`

convhulln *Compute smallest convex hull that encloses a set of points*

Description

Returns an index matrix to the points of simplices ("triangles") that form the smallest convex simplical complex of a set of input points in N-dimensional space. This function interfaces the Qhull library.

Usage

```
convhulln(p, options = "Tv", output.options = NULL,
  return.non.triangulated.facets = FALSE)
```

Arguments

- `p` An M-by-N matrix. The rows of `p` represent M points in N-dimensional space.
- `options` String containing extra options for the underlying Qhull command; see details below and Qhull documentation at ../doc/qhull/html/qconvex.html#synopsis.
- `output.options` String containing Qhull options to control output. Currently `n` (normals) and `FA` (generalised areas and volumes) are supported. Causes an object of return value for details. If `output.options` is `TRUE`, select all supported options.
- `return.non.triangulated.facets` logical defining whether the output facets should be triangulated; `FALSE` by default.

Details

For silent operation, specify the option `Pp`.

Value

If `return.non.triangulated.facets` is `FALSE` (default), return an M-by-N index matrix of which each row defines an N-dimensional "triangle".

If `return.non.triangulated.facets` is `TRUE` then the number of columns equals the maximum number of vertices in a facet, and each row defines a polygon corresponding to a facet of the convex hull with its vertices followed by `NA`s until the end of the row. The indices refer to the rows in `p`.

If the `output.options` or `options` argument contains `FA` or `n`, return a list with class `convhulln` comprising the named elements:

- `hull` The convex hull, represented as a matrix, as described above
- `area` If `FA` is specified, the generalised area of the hull. This is the surface area of a 3D hull or the length of the perimeter of a 2D hull. See ../doc/qhull/html/qh-optf.html#FA.
- `vol` If `FA` is specified, the generalised volume of the hull. This is volume of a 3D hull or the area of a 2D hull. See ../doc/qhull/html/qh-optf.html#FA.

normals If n is specified, this is a matrix hyperplane normals with offsets. See ../doc/qhull/html/qh-opto.html#n.

Note

This is a port of the Octave's (http://www.octave.org) geometry library. The Octave source was written by Kai Habel.

See further notes in delaunayn.

Author(s)

Raoul Grasman, Robert B. Gramacy, Pavlo Mozharovskyi and David Sterratt <david.c.sterratt@ed.ac.uk>

References

Barber, C.B., Dobkin, D.P., and Huhdanpaa, H.T., "The Quickhull algorithm for convex hulls," ACM Trans. on Mathematical Software, *Dec 1996.*

http://www.qhull.org

See Also

convex.hull, delaunayn, surf.tri, distmesh2d, intersectn

Examples

```
# example convhulln
# ==> see also surf.tri to avoid unwanted messages printed to the console by qhull
ps <- matrix(rnorm(3000), ncol=3)  # generate points on a sphere
ps <- sqrt(3)*ps/drop(sqrt((ps^2) %*% rep(1, 3)))
ts.surf <- t(convhulln(ps))  # see the qhull documentations for the options
## Not run:
rgl.triangles(ps[ts.surf,1],ps[ts.surf,2],ps[ts.surf,3],col="blue",alpha=.2)
for(i in 1:(8*360)) rgl.viewpoint(i/8)

## End(Not run)
```

---

delaunayn *Delaunay triangulation in N dimensions*

Description

The Delaunay triangulation is a tessellation of the convex hull of the points such that no N-sphere defined by the N- triangles contains any other points from the set.

Usage

```
delaunayn(p, options = NULL, output.options = NULL, full = FALSE)
```

Arguments

`p`	An M-by-N matrix whose rows represent M points in N-dimensional space.
`options`	String containing extra control options for the underlying Qhull command; see the Qhull documentation (`../doc/qhull/html/qdelaun.html`) for the available options. The `Qbb` option is always passed to Qhull. The default options are `Qcc Qc Qt Qz` for $N < 4$ and `Qcc Qc Qt Qx` for $N >= 4$. If neither of the `QJ` or `Qt` options are supplied, the `Qt` option is passed to Qhull. The `Qt` option ensures all Delaunay regions are simplical (e.g., triangles in 2D). See `../doc/qhull/html/qdelaun.html` for more details. Contrary to the Qhull documentation, no degenerate (zero area) regions are returned with the `Qt` option since the R function removes them from the triangulation. *If* `options` *is specified, the default options are overridden.* It is recommended to use `output.options` for options controlling the outputs.
`output.options`	String containing Qhull options to control output. Currently `Fn` (neighbours) and `Fa` (areas) are supported. Causes an object of return value for details. If `output.options` is `TRUE`, select all supported options.
`full`	Deprecated and will be removed in a future release. Adds options `Fa` and `Fn`.

Value

If neither of the Qhull options `Fn` or `Fa` are specified, return the Delaunay triangulation as a matrix with M rows and $N + 1$ columns in which each row contains a set of indices to the input points `p`. Thus each row describes a simplex of dimension N, e.g. a triangle in 2D or a tetrahedron in 3D.

If the `output.options` argument contains `Fn` or `Fa`, return a list with class `delaunayn` comprising the named elements:

`tri` The Delaunay triangulation described above

`areas` If `Fa` is specified, an M-dimensional vector containing the generalise area of each simplex (e.g. in 2D the areas of triangles; in 3D the volumes of tetrahedra). See `../doc/qhull/html/qh-optf.html#Fa`.

`neighbours` If `Fn` is specified, a list of neighbours of each simplex. See `../doc/qhull/html/qh-optf.html#Fn`

Note

This function interfaces the Qhull library and is a port from Octave (`http://www.octave.org`) to R. Qhull computes convex hulls, Delaunay triangulations, halfspace intersections about a point, Voronoi diagrams, furthest-site Delaunay triangulations, and furthest-site Voronoi diagrams. It runs in 2D, 3D, 4D, and higher dimensions. It implements the Quickhull algorithm for computing the convex hull. Qhull handles round-off errors from floating point arithmetic. It computes volumes, surface areas, and approximations to the convex hull. See the Qhull documentation included in this distribution (the doc directory `../doc/qhull/index.html`).

Qhull does not support constrained Delaunay triangulations, triangulation of non-convex surfaces, mesh generation of non-convex objects, or medium-sized inputs in 9D and higher. A rudimentary algorithm for mesh generation in non-convex regions using Delaunay triangulation is implemented in distmesh2d (currently only 2D).

Author(s)

Raoul Grasman and Robert B. Gramacy; based on the corresponding Octave sources of Kai Habel.

References

Barber, C.B., Dobkin, D.P., and Huhdanpaa, H.T., "The Quickhull algorithm for convex hulls," ACM Trans. on Mathematical Software, *Dec 1996.*

http://www.qhull.org

See Also

tri.mesh, convhulln, surf.tri, distmesh2d

Examples

```
# example delaunayn
d <- c(-1,1)
pc <- as.matrix(rbind(expand.grid(d,d,d),0))
tc <- delaunayn(pc)

# example tetramesh
## Not run:
rgl::rgl.viewpoint(60)
rgl::rgl.light(120,60)
tetramesh(tc,pc, alpha=0.9)

## End(Not run)

tc1 <- delaunayn(pc, output.options="Fa")
## sum of generalised areas is total volume of cube
sum(tc1$areas)
```

---

distmesh2d *A simple mesh generator for non-convex regions*

Description

An unstructured simplex requires a choice of mesh points (vertex nodes) and a triangulation. This is a simple and short algorithm that improves the quality of a mesh by relocating the mesh points according to a relaxation scheme of forces in a truss structure. The topology of the truss is reset using Delaunay triangulation. A (sufficiently smooth) user supplied signed distance function (fd) indicates if a given node is inside or outside the region. Points outside the region are projected back to the boundary.

Usage

```
distmesh2d(fd, fh, h0, bbox, p = NULL, pfix = array(0, dim = c(0, 2)),
  ..., dptol = 0.001, ttol = 0.1, Fscale = 1.2, deltat = 0.2,
  geps = 0.001 * h0, deps = sqrt(.Machine$double.eps) * h0,
  maxiter = 1000, plot = TRUE)
```

Arguments

`fd`	Vectorized signed distance function, for example `mesh.dcircle` or `mesh.diff`, accepting an `n`-by-2 matrix, where `n` is arbitrary, as the first argument.
`fh`	Vectorized function, for example `mesh.hunif`, that returns desired edge length as a function of position. Accepts an `n`-by-2 matrix, where `n` is arbitrary, as its first argument.
`h0`	Initial distance between mesh nodes. (Ignored of `p` is supplied)
`bbox`	Bounding box `cbind(c(xmin,xmax), c(ymin,ymax))`
`p`	An `n`-by-`2` matrix. The rows of `p` represent locations of starting mesh nodes.
`pfix`	`nfix`-by-2 matrix with fixed node positions.
`...`	parameters to be passed to `fd` and/or `fh`
`dptol`	Algorithm stops when all node movements are smaller than `dptol`
`ttol`	Controls how far the points can move (relatively) before a retriangulation with `delaunayn`.
`Fscale`	"Internal pressure" in the edges.
`deltat`	Size of the time step in Euler's method.
`geps`	Tolerance in the geometry evaluations.
`deps`	Stepsize Δx in numerical derivative computation for distance function.
`maxiter`	Maximum iterations.
`plot`	logical. If `TRUE` (default), the mesh is plotted as it is generated.

Details

This is an implementation of original Matlab software of Per-Olof Persson.

Excerpt (modified) from the reference below:

'The algorithm is based on a mechanical analogy between a triangular mesh and a 2D truss structure. In the physical model, the edges of the Delaunay triangles of a set of points correspond to bars of a truss. Each bar has a force-displacement relationship $f(\ell, \ell_0)$ depending on its current length ℓ and its unextended length ℓ_0.'

'External forces on the structure come at the boundaries, on which external forces have normal orientations. These external forces are just large enough to prevent nodes from moving outside the boundary. The position of the nodes are the unknowns, and are found by solving for a static force equilibrium. The hope is that (when `fh = function(p) return(rep(1,nrow(p)))`), the lengths of all the bars at equilibrium will be nearly equal, giving a well-shaped triangular mesh.'

See the references below for all details. Also, see the comments in the source file.

Value

n-by-2 matrix with node positions.

Wishlist

- Implement in C/Fortran
- Implement an nD version as provided in the Matlab package
- Translate other functions of the Matlab package

Author(s)

Raoul Grasman

References

http://persson.berkeley.edu/distmesh/

P.-O. Persson, G. Strang, A Simple Mesh Generator in MATLAB. SIAM Review, Volume 46 (2), pp. 329-345, June 2004

See Also

tri.mesh, delaunayn, mesh.dcircle, mesh.drectangle, mesh.diff, mesh.union, mesh.intersect

Examples

```
# examples distmesh2d
fd <- function(p, ...) sqrt((p^2)%*%c(1,1)) - 1
     # also predefined as `mesh.dcircle'
fh <- function(p,...)  rep(1,nrow(p))
bbox <- matrix(c(-1,1,-1,1),2,2)
p <- distmesh2d(fd,fh,0.2,bbox, maxiter=100)
    # this may take a while:
    # press Esc to get result of current iteration

# example with non-convex region
fd <- function(p, ...) mesh.diff(p , mesh.drectangle, mesh.dcircle, radius=.3)
     # fd defines difference of square and circle

p <- distmesh2d(fd,fh,0.05,bbox,radius=0.3,maxiter=4)
p <- distmesh2d(fd,fh,0.05,bbox,radius=0.3, maxiter=10)
     # continue on previous mesh
```

distmeshnd *A simple mesh generator for non-convex regions in n-D space*

Description

An unstructured simplex requires a choice of mesh points (vertex nodes) and a triangulation. This is a simple and short algorithm that improves the quality of a mesh by relocating the mesh points according to a relaxation scheme of forces in a truss structure. The topology of the truss is reset using Delaunay triangulation. A (sufficiently smooth) user supplied signed distance function (`fd`) indicates if a given node is inside or outside the region. Points outside the region are projected back to the boundary.

Usage

```
distmeshnd(fdist, fh, h, box, pfix = array(dim = c(0, ncol(box))), ...,
  ptol = 0.001, ttol = 0.1, deltat = 0.1, geps = 0.1 * h,
  deps = sqrt(.Machine$double.eps) * h)
```

Arguments

`fdist`	Vectorized signed distance function, for example `mesh.dsphere`, accepting an `m`-by-`n` matrix, where `m` is arbitrary, as the first argument.
`fh`	Vectorized function, for example `mesh.hunif`, that returns desired edge length as a function of position. Accepts an `m`-by-`n` matrix, where `n` is arbitrary, as its first argument.
`h`	Initial distance between mesh nodes.
`box`	2-by-n matrix that specifies the bounding box. (See distmesh2d for an example.)
`pfix`	`nfix`-by-2 matrix with fixed node positions.
`...`	parameters that are passed to `fdist` and `fh`
`ptol`	Algorithm stops when all node movements are smaller than `dptol`
`ttol`	Controls how far the points can move (relatively) before a retriangulation with `delaunayn`.
`deltat`	Size of the time step in Euler's method.
`geps`	Tolerance in the geometry evaluations.
`deps`	Stepsize Δx in numerical derivative computation for distance function.

Details

This is an implementation of original Matlab software of Per-Olof Persson.

Excerpt (modified) from the reference below:

'The algorithm is based on a mechanical analogy between a triangular mesh and a n-D truss structure. In the physical model, the edges of the Delaunay triangles of a set of points correspond to bars of a truss. Each bar has a force-displacement relationship $f(\ell, \ell_0)$ depending on its current length ℓ and its unextended length ℓ_0.'

'External forces on the structure come at the boundaries, on which external forces have normal orientations. These external forces are just large enough to prevent nodes from moving outside the boundary. The position of the nodes are the unknowns, and are found by solving for a static force equilibrium. The hope is that (when `fh = function(p) return(rep(1,nrow(p)))`), the lengths of all the bars at equilibrium will be nearly equal, giving a well-shaped triangular mesh.'

See the references below for all details. Also, see the comments in the source file of `distmesh2d`.

Value

`m`-by-`n` matrix with node positions.

Wishlist

- Implement in C/Fortran
- Translate other functions of the Matlab package

Author(s)

Raoul Grasman; translated from original Matlab sources of Per-Olof Persson.

References

http://persson.berkeley.edu/distmesh/

P.-O. Persson, G. Strang, A Simple Mesh Generator in MATLAB. SIAM Review, Volume 46 (2), pp. 329-345, June 2004

See Also

`distmesh2d`, `tri.mesh`, `delaunayn`, `mesh.dsphere`, `mesh.hunif`, `mesh.diff`, `mesh.union`, `mesh.intersect`

Examples

```
## Not run:
# examples distmeshnd
require(rgl)

fd = function(p, ...) sqrt((p^2)%*%c(1,1,1)) - 1
     # also predefined as `mesh.dsphere'
fh = function(p,...)  rep(1,nrow(p))
     # also predefined as `mesh.hunif'
bbox = matrix(c(-1,1),2,3)
p = distmeshnd(fd,fh,0.2,bbox, maxiter=100)
    # this may take a while:
    # press Esc to get result of current iteration

## End(Not run)
```

dot — *Compute the dot product of two vectors*

Description

If `x` and `y` are matrices, calculate the dot-product along the first non-singleton dimension. If the optional argument `d` is given, calculate the dot-product along this dimension.

Usage

```
dot(x, y, d = NULL)
```

Arguments

`x`	Matrix of vectors
`y`	Matrix of vectors
`d`	Dimension along which to calculate the dot product

Value

Vector with length of `d`th dimension

Author(s)

David Sterratt

entry.value — *Retrieve or set a list of array element values*

Description

`entry.value` retrieves or sets the values in an array `a` at the positions indicated by the rows of a matrix `idx`.

Usage

```
entry.value(a, idx)
```

Arguments

`a`	An array.
`idx`	Numerical matrix with the same number of columns as the number of dimensions of `a`. Each row indices a cell in `a` of which the value is to be retrieved or set.
`value`	An array of length `nrow(idx)`.

Value

`entry.value(a,idx)` returns a vector of values at the indicated cells. `entry.value(a,idx) <- val` changes the indicated cells of `a` to `val`.

Author(s)

Raoul Grasman

Examples

```
a = array(1:(4^4),c(4,4,4,4))
entry.value(a,cbind(1:4,1:4,1:4,1:4))
entry.value(a,cbind(1:4,1:4,1:4,1:4)) <- 0

entry.value(a, as.matrix(expand.grid(1:4,1:4,1:4,1:4)))
     # same as `c(a[1:4,1:4,1:4,1:4])' which is same as `c(a)'
```

---

extprod3d — *Compute external- or 'cross'- product of 3D vectors.*

Description

Computes the external product

$$(x_2y_3 - x_3y_2,\ x_3y_1 - x_1y_3,\ x_1y_2 - x_2y_1)$$

of the 3D vectors in **x** and **y**.

Usage

```
extprod3d(x, y, drop = TRUE)
```

Arguments

`x`	`n`-by-3 matrix. Each row is one **x**-vector
`y`	`n`-by-3 matrix. Each row is one **y**-vector
`drop`	logical. If `TRUE` and if the inputs are one row matrices or vectors, then delete the dimensions of the array returned.

Value

If `n` is greater than 1 or `drop` is `FALSE`, `n`-by-3 matrix; if `n` is 1 and `drop` is `TRUE`, a vector of length 3.

Author(s)

Raoul Grasman

See Also

drop

`feasible.point` *Find point in intersection of convex hulls*

Description

Find point that lies somewhere in interesction of two convex hulls. If such a point does not exist, return NA. The feasible point is found using a linear program similar to the one suggested at ../doc/qhull/html/qhalf.html#notes

Usage

```
feasible.point(ch1, ch2, tol = 0)
```

Arguments

ch1	First convex hull with normals
ch2	Second convex hull with normals
tol	The point must be at least this far within the facets of both convex hulls

---

`halfspacen` *Compute halfspace intersection about a point*

Description

Compute halfspace intersection about a point

Usage

```
halfspacen(p, fp, options = "Tv")
```

Arguments

p	An M-by-$N+1$ matrix. Each row of p represents a halfspace by a N-dimensional normal to a hyperplane and the offset of the hyperplane.
fp	A "feasible" point that is within the space contained within all the halfspaces.
options	String containing extra options, separated by spaces, for the underlying Qhull command; see Qhull documentation at ../doc/qhull/html/qhalf.html.

Value

A N-column matrix containing the intersection points of the hyperplanes ../doc/qhull/html/qhalf.html.

Note

halfspacen was introduced in geometry 0.4.0, and is still under development. It is worth checking results for unexpected behaviour.

Author(s)

David Sterratt

References

Barber, C.B., Dobkin, D.P., and Huhdanpaa, H.T., "The Quickhull algorithm for convex hulls," ACM Trans. on Mathematical Software, *Dec 1996.*

http://www.qhull.org

See Also

convhulln

Examples

```
p <- rbox(0, C=0.5)  # Generate points on a unit cube centered around the origin
ch <- convhulln(p, "n") # Generate convex hull, including normals to facets, with "n" option
# Intersections of half planes
# These points should be the same as the orginal points
pn <- halfspacen(ch$normals, c(0, 0, 0))
```

---

inhulln *Test if points lie in convex hull*

Description

Tests if a set of points lies within a convex hull, returning a boolean vector in which each element is TRUE if the corresponding point lies within the hull and FALSE if it lies outwith the hull or on one of its facets.

Usage

```
inhulln(ch, p)
```

Arguments

ch	Convex hull produced using convhulln
p	An M-by-N matrix of points to test. The rows of p represent M points in N-dimensional space.

Value

A boolean vector with M elements

Note

inhulln was introduced in geometry 0.4.0, and is still under development. It is worth checking results for unexpected behaviour.

Author(s)

David Sterratt

See Also

convhulln

Examples

```
p <- cbind(c(-1, -1, 1), c(-1, 1, -1))
ch <- convhulln(p)
## First point should be in the hull; last two outside
inhulln(ch, rbind(c(-0.5, -0.5),
                  c( 1   ,  1),
                  c(10   ,  0)))

## Test hypercube
p <- rbox(D=4, B=1)
ch <- convhulln(p)
tp <-  cbind(seq(-1.9, 1.9, by=0.2), 0, 0, 0)
pin <- inhulln(ch, tp)
## Points on x-axis should be in box only between -1 and 1
pin == (tp[,1] < 1 & tp[,1] > -1)
```

---

intersectn — *Compute convex hull of intersection of two sets of points*

Description

Compute convex hull of intersection of two sets of points

Usage

```
intersectn(ps1, ps2, tol = 0, return.chs = TRUE, options = "Tv")
```

Arguments

ps1	First set of points
ps2	Second set of points
tol	Tolerance used to determine if a feasible point lies within the convex hulls of both points and to round off the points generated by the halfspace intersection, which sometimes produces points very close together.
return.chs	If TRUE (default) return the convex hulls of the first and second sets of points, as well as the convex hull of the intersection.
options	Options passed to halfspacen. By default this is Tv.

Value

List containing named elements: ch1, the convex hull of the first set of points, with volumes, areas and normals (see convhulln; ch2, the convex hull of the first set of points, with volumes, areas and normals; ps, the intersection points of convex hulls ch1 and ch2; and ch, the convex hull of the intersection points, with volumes, areas and normals.

Note

intersectn was introduced in geometry 0.4.0, and is still under development. It is worth checking results for unexpected behaviour.

Author(s)

David Sterratt

See Also

convhulln, halfspacen, inhulln

Examples

```
# Two overlapping boxes
ps1 <- rbox(0, C=0.5)
ps2 <- rbox(0, C=0.5) + 0.5
out <- intersectn(ps1, ps2)
message("Volume of 1st convex hull: ", out$ch1$vol)
message("Volume of 2nd convex hull: ", out$ch2$vol)
message("Volume of intersection convex hull: ", out$ch$vol)
```

matmax — *Row-wise matrix functions*

Description

Compute maximum or minimum of each row, or sort each row of a matrix, or a set of (equal length) vectors.

Usage

```
matmax(...)
```

Arguments

`...`	A numeric matrix or a set of numeric vectors (that are column-wise bind together into a matrix with `cbind`).

Value

`matmin` and `matmax` return a vector of length `nrow(cbind(...))`. `matsort` returns a matrix of dimension `dim(cbind(...))` with in each row of `cbind(...)` sorted. `matsort(x)` is a lot faster than, e.g., `t(apply(x,1,sort))`, if `x` is tall (i.e., `nrow(x)`»`ncol(x)` and `ncol(x)`<30. If `ncol(x)`>30 then `matsort` simply calls '`t(apply(x,1,sort))`'. `matorder` returns a permutation which rearranges its first argument into ascending order, breaking ties by further arguments.

Author(s)

Raoul Grasman

Examples

```
example(Unique)
```

mesh.dcircle — *Circle distance function*

Description

Signed distance from points p to boundary of circle to allow easy definition of regions in `distmesh2d`.

Usage

```
mesh.dcircle(p, radius = 1, ...)
```

Arguments

`p`	A matrix with 2 columns (3 in `mesh.dsphere`), each row representing a point in the plane.
`radius`	radius of circle
`...`	additional arguments (not used)

Value

A vector of length `nrow(p)` containing the signed distances to the circle

Author(s)

Raoul Grasman; translated from original Matlab sources of Per-Olof Persson.

References

http://persson.berkeley.edu/distmesh/

P.-O. Persson, G. Strang, A Simple Mesh Generator in MATLAB. SIAM Review, Volume 46 (2), pp. 329-345, June 2004

See Also

`distmesh2d`, `mesh.drectangle`, `mesh.diff`, `mesh.intersect`, `mesh.union`

Examples

```
example(distmesh2d)
```

---

mesh.diff — *Difference, union and intersection operation on two regions*

Description

Compute the signed distances from points `p` to a region defined by the difference, union or intersection of regions specified by the functions `regionA` and `regionB`. `regionA` and `regionB` must accept a matrix `p` with 2 columns as their first argument, and must return a vector of length `nrow(p)` containing the signed distances of the supplied points in `p` to their respective regions.

Usage

```
mesh.diff(p, regionA, regionB, ...)
```

Arguments

`p`	A matrix with 2 columns (3 in `mesh.dsphere`), each row representing a point in the plane.
`regionA`	vectorized function describing region A in the union / intersection / difference
`regionB`	vectorized function describing region B in the union / intersection / difference
`...`	additional arguments passed to `regionA` and `regionB`

Value

A vector of length `nrow(p)` containing the signed distances to the boundary of the region.

Author(s)

Raoul Grasman; translated from original Matlab sources of Per-Olof Persson.

See Also

`distmesh2d`, `mesh.dcircle`, `mesh.drectangle` `mesh.dsphere`

`mesh.drectangle` *Rectangle distance function*

Description

Signed distance from points `p` to boundary of rectangle to allow easy definition of regions in `distmesh2d`.

Usage

```
mesh.drectangle(p, x1 = -1/2, y1 = -1/2, x2 = 1/2, y2 = 1/2, ...)
```

Arguments

`p`	A matrix with 2 columns, each row representing a point in the plane.
`x1`	lower left corner of rectangle
`y1`	lower left corner of rectangle
`x2`	upper right corner of rectangle
`y2`	upper right corner of rectangle
`...`	additional arguments (not used)

Value

a vector of length `nrow(p)` containing the signed distances

Author(s)

Raoul Grasman; translated from original Matlab sources of Per-Olof Persson.

References

http://persson.berkeley.edu/distmesh/

P.-O. Persson, G. Strang, A Simple Mesh Generator in MATLAB. SIAM Review, Volume 46 (2), pp. 329-345, June 2004

See Also

`distmesh2d`, `mesh.drectangle`, `mesh.diff`, `mesh.intersect`, `mesh.union`

Examples

```
example(distmesh2d)
```

---

mesh.dsphere *Sphere distance function*

Description

Signed distance from points `p` to boundary of sphere to allow easy definition of regions in `distmeshnd`.

Usage

```
mesh.dsphere(p, radius = 1, ...)
```

Arguments

`p`	A matrix with 2 columns (3 in `mesh.dsphere`), each row representing a point in the plane.
`radius`	radius of sphere
`...`	additional arguments (not used)

Value

A vector of length `nrow(p)` containing the signed distances to the sphere

Author(s)

Raoul Grasman; translated from original Matlab sources of Per-Olof Persson.

References

http://persson.berkeley.edu/distmesh/

P.-O. Persson, G. Strang, A Simple Mesh Generator in MATLAB. SIAM Review, Volume 46 (2), pp. 329-345, June 2004

See Also

`distmeshnd`

Examples

```
example(distmeshnd)
```

`mesh.hunif` *Uniform desired edge length*

Description

Uniform desired edge length function of position to allow easy definition of regions when passed as the `fh` argument of `distmesh2d` or `distmeshnd`.

Usage

```
mesh.hunif(p, ...)
```

Arguments

`p`	A `n`-by-`m` matrix, each row representing a point in an `m`-dimensional space.
`...`	additional arguments (not used)

Value

Vector of ones of length `n`.

Author(s)

Raoul Grasman; translated from original Matlab sources of Per-Olof Persson.

See Also

`distmesh2d` and `distmeshnd`.

pol2cart *Transform polar or cylindrical coordinates to Cartesian coordinates.*

Description

The inputs `theta`, `r`, (and `z`) must be the same shape, or scalar. If called with a single matrix argument then each row of `P` represents the polar/(cylindrical) coordinate (`theta`, `r` (, `z`)).

Usage

```
pol2cart(theta, r = NULL, z = NULL)
```

Arguments

`theta`	describes the angle relative to the positive x-axis.
`r`	is the distance to the z-axis (0, 0, z).
`z`	(optional) is the z-coordinate

Value

a matrix `C` where each row represents one Cartesian coordinate (`x`, `y` (, `z`)).

Author(s)

Kai Habel

David Sterratt

See Also

`cart2pol`, `sph2cart`, `cart2sph`

polyarea *Determines area of a polygon by triangle method.*

Description

Determines area of a polygon by triangle method. The variables `x` and `y` define the vertex pairs, and must therefore have the same shape. They can be either vectors or arrays. If they are arrays then the columns of `x` and `y` are treated separately and an area returned for each.

Usage

```
polyarea(x, y, d = 1)
```

Arguments

`x`	X coordinates of vertices.
`y`	Y coordinates of vertices.
`d`	Dimension of array to work along.

Details

If the optional `dim` argument is given, then `polyarea` works along this dimension of the arrays `x` and `y`.

Value

Area(s) of polygon(s).

Author(s)

David Sterratt based on the octave sources by David M. Doolin

Examples

```
x <- c(1, 1, 3, 3, 1)
y <- c(1, 3, 3, 1, 1)
polyarea(x, y)
polyarea(cbind(x, x), cbind(y, y)) ##  c(4, 4)
polyarea(cbind(x, x), cbind(y, y), 1) ##  c(4, 4)
polyarea(rbind(x, x), rbind(y, y), 2) ##  c(4, 4)
```

`rbox` *Generate various point distributions*

Description

Default is corners of a hypercube.

Usage

```
rbox(n = 3000, D = 3, B = 0.5, C = NA)
```

Arguments

`n`	number of random points in hypercube
`D`	number of dimensions of hypercube
`B`	bounding box coordinate - faces will be `-B` and `B` from origin
`C`	add a unit hypercube to the output - faces will be `-C` and `C` from origin

Value

Matrix of points

Author(s)

David Sterratt

`sph2cart` *Transform spherical coordinates to Cartesian coordinates*

Description

The inputs `theta`, `phi`, and `r` must be the same shape, or scalar. If called with a single matrix argument then each row of `S` represents the spherical coordinate (`theta`, `phi`, `r`).

Usage

```
sph2cart(theta, phi = NULL, r = NULL)
```

Arguments

`theta`	describes the angle relative to the positive x-axis.
`phi`	is the angle relative to the xy-plane.
`r`	is the distance to the origin `(0, 0, 0)`. If only a single return argument is requested then return a matrix `C` where each row represents one Cartesian coordinate (`x`, `y`, `z`).

Author(s)

Kai Habel

David Sterratt

See Also

`cart2sph`, `pol2cart`, `cart2pol`

surf.tri	*Find surface triangles from tetrahedral mesh*

Description

Find surface triangles from tetrahedral mesh typically obtained with delaunayn.

Usage

```
surf.tri(p, t)
```

Arguments

p	An n-by-3 matrix. The rows of p represent n points in dim-dimensional space.
t	Matrix with 4 columns, interpreted as output of delaunayn.

Details

surf.tri and convhulln serve a similar purpose in 3D, but surf.tri also works for non-convex meshes obtained e.g. with distmeshnd. It also does not produce currently unavoidable diagnostic output on the console as convhulln does at the Rterm console–i.e., surf.tri is silent.

Value

An m-by-3 index matrix of which each row defines a triangle. The indices refer to the rows in p.

Note

surf.tri was based on Matlab code for mesh of Per-Olof Persson (http://persson.berkeley.edu/distmesh/).

Author(s)

Raoul Grasman

See Also

tri.mesh, convhulln, surf.tri, distmesh2d

Examples

```
## Not run:
# more extensive example of surf.tri

# url's of publically available data:
data1.url = "http://neuroimage.usc.edu/USCPhantom/mesh_data.bin"
data2.url = "http://neuroimage.usc.edu/USCPhantom/CT_PCS_trans.bin"
```

```
meshdata = R.matlab::readMat(url(data1.url))
elec = R.matlab::readMat(url(data2.url))$eeg.ct2pcs/1000
brain = meshdata$mesh.brain[,c(1,3,2)]
scalp = meshdata$mesh.scalp[,c(1,3,2)]
skull = meshdata$mesh.skull[,c(1,3,2)]
tbr = t(surf.tri(brain, delaunayn(brain)))
tsk = t(surf.tri(skull, delaunayn(skull)))
tsc = t(surf.tri(scalp, delaunayn(scalp)))
rgl::rgl.triangles(brain[tbr,1], brain[tbr,2], brain[tbr,3],col="gray")
rgl::rgl.triangles(skull[tsk,1], skull[tsk,2], skull[tsk,3],col="white", alpha=0.3)
rgl::rgl.triangles(scalp[tsc,1], scalp[tsc,2], scalp[tsc,3],col="#a53900", alpha=0.6)
rgl::rgl.viewpoint(-40,30,.4,zoom=.03)
lx = c(-.025,.025); ly = -c(.02,.02);
rgl::rgl.spheres(elec[,1],elec[,3],elec[,2],radius=.0025,col='gray')
rgl::rgl.spheres( lx, ly,.11,radius=.015,col="white")
rgl::rgl.spheres( lx, ly,.116,radius=.015*.7,col="brown")
rgl::rgl.spheres( lx, ly,.124,radius=.015*.25,col="black")

## End(Not run)
```

`tetramesh` *Render tetrahedron mesh (3D)*

Description

`tetramesh(T, X, col)` uses the rgl package to display the tetrahedrons defined in the m-by-4 matrix T as mesh. Each row of `T` specifies a tetrahedron by giving the 4 indices of its points in `X`.

Usage

```
tetramesh(T, X, col = grDevices::heat.colors(nrow(T)), clear = TRUE,
  ...)
```

Arguments

`T`	T is a `m`-by-3 matrix in trimesh and `m`-by-4 in tetramesh. A row of `T` contains indices into `X` of the vertices of a triangle/tetrahedron. `T` is usually the output of delaunayn.
`X`	X is an n-by-2/n-by-3 matrix. The rows of X represent `n` points in 2D/3D space.
`col`	The tetrahedron colour. See rgl documentation for details.
`clear`	Should the current rendering device be cleared?
`...`	Parameters to the rendering device. See the rgl package.

Author(s)

Raoul Grasman

See Also

trimesh, rgl, delaunayn, convhulln, surf.tri

Examples

```
## Not run:
# example delaunayn
d = c(-1,1)
pc = as.matrix(rbind(expand.grid(d,d,d),0))
tc = delaunayn(pc)

# example tetramesh
clr = rep(1,3) %o% (1:nrow(tc)+1)
rgl::rgl.viewpoint(60,fov=20)
rgl::rgl.light(270,60)
tetramesh(tc,pc,alpha=0.7,col=clr)

## End(Not run)
```

`to.mesh3d` *Convert convhulln object to RGL mesh*

Description

Convert convhulln object to RGL mesh

Usage

```
to.mesh3d(x, ...)
```

Arguments

x	convhulln object
...	Arguments to qmesh3d or tmesh3d

Value

mesh3d object, which can be displayed in RGL with dot3d, wire3d or shade3d

See Also

as.mesh3d

trimesh — *Display triangles mesh (2D)*

Description

`trimesh(T, p)` displays the triangles defined in the m-by-3 matrix `T` and points `p` as a mesh. Each row of `T` specifies a triangle by giving the 3 indices of its points in `X`.

Usage

```
trimesh(T, p, p2, add = FALSE, axis = FALSE, boxed = FALSE, ...)
```

Arguments

`T`	T is a `m`-by-3 matrix. A row of `T` contains indices into `X` of the vertices of a triangle. `T` is usually the output of `delaunayn`.
`p`	A vector or a matrix.
`p2`	if `p` is not a matrix `p` and `p2` are bind to a matrix with `cbind`.
`add`	Add to existing plot in current active device?
`axis`	Draw axes?
`boxed`	Plot box?
`...`	Parameters to the rendering device. See the rgl package.

Author(s)

Raoul Grasman

See Also

`tetramesh`, `rgl`, `delaunayn`, `convhulln`, `surf.tri`

Examples

```
#example trimesh
p = cbind(x=rnorm(30), y=rnorm(30))
tt = delaunayn(p)
trimesh(tt,p)
```

tsearch — *Search for the enclosing Delaunay convex hull*

Description

For `t <- delaunay(cbind(x, y))`, where `(x, y)` is a 2D set of points, `tsearch(x, y, t, xi, yi)` finds the index in `t` containing the points `(xi, yi)`. For points outside the convex hull the index is `NA`.

Usage

```
tsearch(x, y, t, xi, yi, bary = FALSE, method = "quadtree")
```

Arguments

`x`	X-coordinates of triangulation points
`y`	Y-coordinates of triangulation points
`t`	Triangulation, e.g. produced by `t <-delaunayn(cbind(x, y))`
`xi`	X-coordinates of points to test
`yi`	Y-coordinates of points to test
`bary`	If `TRUE` return barycentric coordinates as well as index of triangle.
`method`	One of `"quadtree"` or `"orig"`. The Quadtree algorithm is much faster and new from version 0.4.0. The `orig` option uses the tsearch algorithm adapted from Octave code. Its use is deprecated and it may be removed from a future version of the package.

Value

If `bary` is `FALSE`, the index in `t` containing the points `(xi, yi)`. For points outside the convex hull the index is `NA`. If `bary` is `TRUE`, a list containing:

list("idx") the index in `t` containing the points `(xi, yi)`

list("p") a 3-column matrix containing the barycentric coordinates with respect to the enclosing triangle of each point `(xi, yi)`.

Note

The original Octave function is Copyright (C) 2007-2012 David Bateman

Author(s)

Jean-Romain Roussel (Quadtree algorithm), David Sterratt (Octave-based implementation)

See Also

`tsearchn`, `delaunayn`

`tsearchn` *Search for the enclosing Delaunay convex hull*

Description

For `t = delaunayn(x)`, where `x` is a set of points in N dimensions, `tsearchn(x, t, xi)` finds the index in `t` containing the points `xi`. For points outside the convex hull, `idx` is `NA`. `tsearchn` also returns the barycentric coordinates `p` of the enclosing triangles.

Usage

```
tsearchn(x, t, xi, ...)
```

Arguments

`x`	An N-column matrix, in which each row represents a point in N-dimensional space.
`t`	A matrix with $N+1$ columns. A row of `t` contains indices into `x` of the vertices of an N-dimensional simplex. `t` is usually the output of delaunayn.
`xi`	An M-by-N matrix. The rows of `xi` represent M points in N-dimensional space whose positions in the mesh are being sought.
`...`	Additional arguments

Details

If `x` is `NA` and the `t` is a `delaunayn` object produced by `delaunayn` with the `full` option, then use the Qhull library to perform the search. Please note that this is experimental in geometry version 0.4.0 and is only partly tested for 3D hulls, and does not yet work for hulls of 4 dimensions and above.

Value

A list containing:

`idx` An M-long vector containing the indices of the row of `t` in which each point in `xi` is found.

`p` An M-by-$N+1$ matrix containing the barycentric coordinates with respect to the enclosing simplex of each point in `xi`.

Note

Based on the Octave function Copyright (C) 2007-2012 David Bateman.

Author(s)

David Sterratt

See Also

tsearch, delaunayn

Unique *Extract Unique Rows*

Description

'Unique' returns a vector, data frame or array like 'x' but with duplicate elements removed.

Usage

```
Unique(X, rows.are.sets = FALSE)
```

Arguments

X	Numerical matrix.
rows.are.sets	If 'TRUE', rows are treated as sets - i.e., to define uniqueness, the order of the rows does not matter.

Value

Matrix of the same number of columns as x, with the unique rows in x sorted according to the columns of x. If rows.are.sets = TRUE the rows are also sorted.

Note

'Unique' is (under circumstances) much quicker than the more generic base function 'unique'.

Author(s)

Raoul Grasman

Examples

```
# `Unique' is faster than `unique'
x = matrix(sample(1:(4*8),4*8),ncol=4)
y = x[sample(1:nrow(x),3000,TRUE), ]
gc(); system.time(unique(y))
gc(); system.time(Unique(y))

#
z = Unique(y)
x[matorder(x),]
z[matorder(z),]
```

Index