

Package ‘geometry’

January 2, 2012

Title Mesh generation and surface tessellation

Version 0.2-0

Date 2011-08-24

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

Description This package makes the qhull library (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 2-d, 3-d, 4-d, 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. Currently the package only gives access to Delaunay triangulation and convex hull computation.

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

Depends R (>= 2.5.0)

Suggests rgl, R.matlab, tripack

URL <http://geometry.r-forge.r-project.org/>

License GPL (>= 2) + file LICENSE

Repository CRAN

Repository/R-Forge/Project geometry

Repository/R-Forge/Revision 23

Date/Publication 2011-09-14 15:14:38

R topics documented:

convhulln	2
delaunayn	4
distmesh2d	6
distmeshnd	8
entry.value	10
extprod3d	11
matmax, matmin, matsort	12
mesh.functions	13
surf.tri	14
tetramesh	15
tsearch	17
tsearchn	18
Unique	19
Index	20

convhulln	<i>Compute smallest convex hull that encloses a set of points</i>
-----------	---

Description

Returns an index matrix to the points of simplices (“triangles”) that form the smallest convex simplicial complex of a set of input points in N-dimensional space.

Usage

```
convhulln(p, options = "Tv")
```

Arguments

p	An n-by-dim matrix. The rows of p represent n points in dim-dimensional space.
options	Optional options, see details below and Qhull documentation.

Details

This function interfaces the qhull library, and intends to be a port from Octave to R.

The input n-by-dim matrix contains n points of dimension dim. If a second optional argument is given, it must be a string containing extra options for the underlying qhull command. The options always include "Qt". (See the Qhull documentation for the available options - refer to [delaunayn](#).)

Value

An m-by-dim index matrix of which each row defines a dim-dimensional “triangle”. The indices refer to the rows in p. If the option "FA" is provided, then the output is a list with entries \$hull containing the matrix mentioned above, and \$area and \$vol with the area and volume of the hull described by the matrix.

Note

This intends to be a port of the Octave's (<http://www.octave.org>) geometry library. The sources originals were from Kai Habel.

All console printing is sent to a file in the CWD called "qhull_out.txt" unless another file is specified with the TO option – see the qhull documentation. To get the usual progress-related output specify the R-specific option "Pp Ps". The option "FA" results in the area and volume of the convex hull to be included in the output list.

See further notes in [delaunayn](#).

Author(s)

Raoul Grasman and Robert B. Gramacy <bobby@statslab.cam.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](#)

Examples

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

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

## End(Not run)

# example convhulln
# ==> see also surf.tri to avoid unwanted messages printed to the console by qhull
ps = matrix(rnorm(3000),ncol=3) # generate point on a sphere
ps = sqrt(3) * ps / drop(sqrt((ps^2) %*% rep(1,3)))
ts.surf = t( convhulln(ps,"QJ") ) # 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)
```

`del aunayn`*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

```
del aunayn(p, options = "QJ")
```

Arguments

<code>p</code>	<code>p</code> is an n -by- dim matrix. The rows of <code>p</code> represent n points in dim -dimensional space.
<code>options</code>	Optional options, see details below.

Details

This function interfaces the `qhull` library, and intends to be a port from Octave 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 2-d, 3-d, 4-d, and higher dimensions. It implements the Quickhull algorithm for computing the convex hull. `Qhull` handles roundoff 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/index.htm`).

The input n -by- dim matrix contains n points of dimension dim . The return matrix `T` has m rows and $dim+1$ columns. It contains for each row a set of indices to the points, which describes a simplex of dimension dim . The 3D simplex is a tetrahedron.

If a second optional argument is given, it must be a string containing extra options for the underlying `qhull` command. In particular, "Qt" may be useful for juggling the input to cope with non-simplicial cases. (See the `Qhull` documentation (`../doc/index.htm`) for the available options.)

Value

The return matrix has m rows and $dim+1$ columns. It contains for each row a set of indices to the points, which describes a simplex of dimension dim .

Note

This intends to be a port of the Octave's (<http://www.octave.org>) geometry library. The sources originals were from Kai Habel.

The current implementation calls `Qhull` always with the "QJ" option. (See `Qhull` documentation for details).

All console printing is sent to a file in the CWD called “qhull_out.txt” unless another file is specified with the TO option – see the qhull documentation. To get the usual progress-related output specify the R-specific option "Pp Ps".

Qhull does not support constrained Delaunay triangulations, triangulation of non-convex surfaces, mesh generation of non-convex objects, or medium-sized inputs in 9-D 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 <bobby@statslab.cam.ac.uk>; 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:
library(rgl)
rgl.viewpoint(60)
rgl.light(120,60)
tetramesh(tc,pc, alpha=0.9)

## End(Not run)

# example surf.tri
# ==> see also convhulln, but it currently prints an unavoidable
# message to the console
ps = matrix(rnorm(3000),ncol=3) # generate pointst on a sphere
ps = sqrt(3) * ps / drop(sqrt((ps^2) %*%rep(1,3)))
ts = delaunayn(ps)
ts.surf = t( surf.tri(ps,ts) )
## 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)
```

distmesh2d

A simple mesh generator for non-convex regions

Description

An unstructured simplex requires a choice of meshpoints (vertex nodes) and a triangulation. This is a simple and short algorithm that improves the quality of a mesh by relocating the meshpoints 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)
```

Arguments

fd	Vectorized signed distance function, accepting an n-by-2 matrix, where n is arbitrary, as the first argument.
fh	Vectorized function 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 if p is supplied)
bbox	Bounding box <code>cbind(c(xmin,xmax), c(ymin,ymax))</code>
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 Eulers method.
geps	Tolerance in the geometry evaluations.
deps	Stepsize Δx in numerical derivative computation for distance function.
maxiter	Maximum iterations.

Details

This is an R 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://www-math.mit.edu/~persson/mesh/>

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 meshpoints (vertex nodes) and a triangulation. This is a simple and short algorithm that improves the quality of a mesh by relocating the meshpoints 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, accepting an m-by-n matrix, where m is arbitrary, as the first argument.
fh	Vectorized function 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 Eulers method.
geps	Tolerance in the geometry evaluations.
deps	Stepsize Δx in numerical derivative computation for distance function.

Details

This is an R 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://www-math.mit.edu/~persson/mesh/>

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`, `de launay`, `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

# 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

## End(Not run)

```

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)
entry.value(a, idx) <- value

```

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)
```

Arguments

x	n-by-3 matrix. Each row is one x -vector
y	n-by-3 matrix. Each row is one y -vector

Value

n-by-3 matrix

Author(s)

Raoul Grasman

matmax, matmin, matsort

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(...)

matmin(...)

matsort(...)

matorder(...)

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.functions *Special Distance Functions*

Description

Elementary distance functions, usefull for defining distance functions of more complex regions.

Usage

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

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

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

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

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

Arguments

p	A matrix with 2 columns (3 in mesh.dsphere), each row representing a point in the plane.
radius	radius of circle
x1	lower left corner of rectangle
y1	lower left corner of rectangle
x2	upper right corner of rectangle
y2	upper right corner of rectangle
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

Details

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.

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://www-math.mit.edu/~persson/mesh/>

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

See Also

[distmesh2d](#)

Examples

```
example(distmesh2d)
```

surf.tri

Find surface triangles from tetrahedra mesh

Description

Find surface triangles from tetrahedron mesh typically obtained with [de launayn](#).

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 [de launayn](#).

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://www-math.mit.edu/~persson/mesh/index.html>).

Author(s)

Raoul Grasman

See Also[tri.mesh](#), [convhulln](#), [surf.tri](#), [distmesh2d](#)**Examples**

```
## Not run:
# more extensive example of surf.tri
library(rgl)          # to render tessellation
library(R.matlab)    # to read matlab .mat files

# 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 = readMat(url(data1.url))
elec = 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.triangles(brain[tbr,1], brain[tbr,2], brain[tbr,3],col="gray")
rgl.triangles(skull[tsk,1], skull[tsk,2], skull[tsk,3],col="white", alpha=0.3)
rgl.triangles(scalp[tsc,1], scalp[tsc,2], scalp[tsc,3],col="#a53900", alpha=0.6)
rgl.viewpoint(-40,30,.4,zoom=.03)
lx = c(-.025,.025); ly = -c(.02,.02);
rgl.spheres(elec[,1],elec[,3],elec[,2],radius=.0025,col='gray')
rgl.spheres( lx, ly,.11,radius=.015,col="white")
rgl.spheres( lx, ly,.116,radius=.015*.7,col="brown")
rgl.spheres( lx, ly,.124,radius=.015*.25,col="black")

## End(Not run)
```

tetramesh

*Display triangles mesh (2D) 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.

Usage

```
tetramesh(T, X, col = heat.colors(nrow(T)), clear = TRUE, ...)
trimesh(T, p, p2, add = FALSE, axis = FALSE, boxed = FALSE, ...)
```

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.
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?
col	The tetrahedron color. See rgl documentation for details.
clear	Should the current rendering device be cleared?
...	Parameters to the rendering device. See the rgl package.

Details

Each row of T specifies a triangle by giving the 3 indices of its points in X. Each row of T specifies a tetrahedron by giving the 4 indices of its points in X.

Author(s)

Raoul Grasman

See Also

[rgl](#), [delaunayn](#), [convhulln](#), [surf.tri](#)

Examples

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

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

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

## End(Not run)
```

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)
```

Arguments

<code>x</code>	X-coordinates of triangulation points
<code>y</code>	Y-coordinates of triangulation points
<code>t</code>	Triangulation, e.g. produced by <code>t = delaunayn(cbind(x, y))</code>
<code>xi</code>	X-coordinates of points to test
<code>yi</code>	Y-coordinates of points to test
<code>bary</code>	If TRUE return barycentric coordinates as well as index of triangle.

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:

<code>idx</code>	the index in <code>t</code> containing the points <code>(xi, yi)</code>
<code>p</code>	a 3-column matrix containing the barycentric coordinates with respect to the enclosing triangle of each point <code>code(xi, yi)</code> .

Author(s)

David Sterratt

See Also

`tsearchn`, `delaunayn`

tsearchn

Search for the enclosing Delaunay convex hull

Description

For $t = \text{delaunayn}(x)$, where x is a set of points in d dimensions, $\text{tsearchn}(x, t, xi)$ 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, fast=TRUE)
```

Arguments

x	An n -by- d matrix. The rows of x represent n points in d -dimensional space.
t	A m -by- $d+1$ matrix. A row of t contains indices into x of the vertices of a d -dimensional simplex. t is usually the output of delaunayn .
xi	An n_i -by- d matrix. The rows of xi represent n points in d -dimensional space whose positions in the mesh are being sought.
$fast$	If the data is in 2D, use the fast C-based tsearch function to produce the results.

Value

A list containing:

idx	An n_i -long vector containing the indices of the row of t in which each point in xi is found.
p	An n_i -by- $d+1$ matrix containing the barycentric coordinates with respect to the enclosing simplex of each point in xi .

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

- *Topic **arith**
 - entry.value, 10
 - extprod3d, 11
 - matmax, matmin, matsort, 12
 - mesh.functions, 13
 - Unique, 19
- *Topic **array**
 - entry.value, 10
 - extprod3d, 11
 - matmax, matmin, matsort, 12
 - Unique, 19
- *Topic **dplot**
 - convhulln, 2
 - delaunayn, 4
 - distmesh2d, 6
 - distmeshnd, 8
 - surf.tri, 14
- *Topic **graphs**
 - convhulln, 2
 - delaunayn, 4
 - distmesh2d, 6
 - distmeshnd, 8
- *Topic **hplot**
 - tetramesh, 15
- *Topic **math**
 - convhulln, 2
 - delaunayn, 4
 - distmesh2d, 6
 - distmeshnd, 8
 - entry.value, 10
 - extprod3d, 11
 - mesh.functions, 13
 - surf.tri, 14
 - Unique, 19
- *Topic **optimize**
 - distmesh2d, 6
 - distmeshnd, 8
 - surf.tri, 14
- convex.hull, 3
- convhulln, 2, 5, 14–16
- delaunayn, 2, 3, 4, 6–9, 14, 16
- distmesh2d, 3, 5, 6, 8, 9, 14, 15
- distmeshnd, 8, 14
- entry.value, 10
- entry.value<- (entry.value), 10
- extprod3d, 11
- matmax (matmax, matmin, matsort), 12
- matmax, matmin, matsort, 12
- matmin (matmax, matmin, matsort), 12
- matorder (matmax, matmin, matsort), 12
- matsort (matmax, matmin, matsort), 12
- mesh.dcircle, 7
- mesh.dcircle (mesh.functions), 13
- mesh.diff, 7, 9
- mesh.diff (mesh.functions), 13
- mesh.drectangle, 7
- mesh.drectangle (mesh.functions), 13
- mesh.dsphere, 9
- mesh.dsphere (mesh.functions), 13
- mesh.functions, 13
- mesh.hunif, 9
- mesh.hunif (mesh.functions), 13
- mesh.intersect, 7, 9
- mesh.intersect (mesh.functions), 13
- mesh.union, 7, 9
- mesh.union (mesh.functions), 13
- rgl, 15, 16
- surf.tri, 3, 5, 14, 15, 16
- tetramesh, 15
- tri.mesh, 5, 7, 9, 15
- trimesh (tetramesh), 15
- tsearch, 17
- tsearchn, 18
- Unique, 19