Package 'deldir'

October 13, 2022

Version 1.0-6

Date 2021-10-23

Title Delaunay Triangulation and Dirichlet (Voronoi) Tessellation

Author Rolf Turner

Maintainer Rolf Turner <r.turner@auckland.ac.nz>

Depends R (>= 3.5.0)

Suggests polyclip

Imports graphics, grDevices

Description Calculates the Delaunay triangulation and the Dirichlet or Voronoi tessellation (with respect to the entire plane) of a planar point set. Plots triangulations and tessellations in various ways. Clips tessellations to sub-windows. Calculates perimeters of tessellations. Summarises information about the tiles of the tessellation. Calculates the centroidal Voronoi (Dirichlet) tessellation using Lloyd's algorithm.

LazyData true

ByteCompile true

License GPL (>= 2)

NeedsCompilation yes

Repository CRAN

Date/Publication 2021-10-23 04:50:03 UTC

R topics documented:

`cvt` *Centroidal Voronoi (Dirichlet) tessellation.*

Description

Calculates the centroidal Voronoi (Dirichlet) tessellation using Lloyd's algorithm.

Usage

```
cvt(object, stopcrit = c("change", "maxit"), tol = NULL,
   maxit = 100, verbose = FALSE)
```

Arguments

`object`	An object of class either `"deldir"` (as returned by `deldir()`) or `"tile.list"` (as returned by `tile.list()`).
`stopcrit`	Text string specifying the stopping criterion for the algorithm. If this is `"change"` then the algorithm halts when the maximum change in in the distances between corresponding centroids, between iterations, is less than `tol` (see below). It `stopcrit` is `"maxit"` then the algorithm halts after a specified number of iterations (`maxit`; see below) have been completed. This argument may be abbreviated, e.g. to `"c"` or `"m"`.
`tol`	The tolerance used when the stopping criterion is `"change"`. Defaults to `.Machine$double.eps`.
`maxit`	The maximum number of iterations to perform when the stopping criterion is `"maxit"`.
`verbose`	Logical scalar. If `verbose` is `TRUE` then rudimentary "progress reports" are printed out, every 10 iterations if the stopping criterion is `"change"` or every iteration if the stopping criterion is `"maxit"`.

Details

The algorithm iteratively tessellates a set of points and then replaces the points with the centroids of the tiles associated with those points. "Eventually" (at convergence) points and the centroids of their associated tiles coincide.

Value

A list with components:

`centroids`	A data frame with columns `"x"` and `"y"` specifying the coordinates of the limiting locations of the tile centroids.
`tiles`	An object of class `"tile.list"` specifying the Dirichlet (Voronoi) tiles in the tessellation of the points whose coordinates are given in `centroids`. Note the tile associated with the ith point has centroid *equal* to that point.

Note

This function was added to the `deldir` package at the suggestion of Dr. Micha\"el Aupetit, Senior Scientist at the Qatar Computing Research Institute, Hamad Bin Khalifa University.

Author(s)

Rolf Turner <r.turner@auckland.ac.nz>

References

https://en.wikipedia.org/wiki/Lloyd's_algorithm

Lloyd, Stuart P. (1982). Least squares quantization in PCM. *IEEE Transactions on Information Theory* **28** (2), pp. 129–137, doi:10.1109/TIT.1982.1056489.

See Also

`deldir()` `tile.list()`

Examples

```
 # Takes too long.
    set.seed(42)
    x <- runif(20)
    y <- runif(20)
    dxy <- deldir(x,y,rw=c(0,1,0,1))
    cxy1 <- cvt(dxy,verb=TRUE)
    plot(cxy1$tiles)
    with(cxy1$centroids,points(x,y,pch=20,col="red"))
    cxy2 <- cvt(dxy,stopcrit="m",verb=TRUE)
    plot(cxy2$tiles)
    with(cxy2$centroids,points(x,y,pch=20,col="red"))
# Visually indistinguishable from the cxy1 plot.
# But ...
    all.equal(cxy1$centroids,cxy2$centroids) # Not quite.
```

```
    cxy3 <- cvt(dxy,stopcrit="m",verb=TRUE,maxit=250)
    all.equal(cxy1$centroids,cxy3$centroids) # Close, but no cigar.
    cxy4 <- cvt(dxy,verb=TRUE,tol=1e-14)
    cxy5 <- cvt(dxy,stopcrit="m",verb=TRUE,maxit=600)
    all.equal(cxy4$centroids,cxy5$centroids) # TRUE
# Takes a lot of iterations or a really small tolerance
# to get "good" convergence.  But this is almost surely
# of no practical importance.
    txy <- tile.list(dxy)
    cxy6 <- cvt(txy)
    all.equal(cxy6$centroids,cxy1$centroids) # TRUE
```

`deldir` *Delaunay triangulation and Dirichlet tessellation*

Description

This function computes the Delaunay triangulation (and hence the Dirichlet or Voronoi tessellation) of a planar point set according to the second (iterative) algorithm of Lee and Schacter — see **References**. The triangulation is made to be with respect to the whole plane by "suspending" it from so-called ideal points (-Inf,-Inf), (Inf,-Inf) (Inf,Inf), and (-Inf,Inf). The triangulation is also enclosed in a finite rectangular window.

Usage

```
deldir(x, y, z=NULL, rw=NULL, eps=1e-09, sort=TRUE, plot=FALSE,
       round=TRUE,digits=6, ...)
```

Arguments

`x,y`	These arguments specify the coordinates of the point set being triangulated/tessellated. Argument `x` may be a numeric vector or it may be a data structure consisting of a matrix, a data frame, a generic list, or an object of class `"ppp"`. (See package `spatstat`.) Argument `y`, if specified, is always a numeric vector. The `"x"` and `"y"` coordinates are extracted from arguments `x` and `y` according to (what can be, in some instances) a rather complicated protocol. See **Notes on extracting the coordinates** for details of this protocol.
`z`	Optional argument specifying "auxiliary" values or "tags" associated with the respective points. (See **Notes on "tags"**.) This argument may be a vector or factor whose entries constitute these tags, or it may be a text string naming such a vector or factor. If `z`, or the object named by `z` is a vector (rather than a factor) it may be of any mode (numeric, character, logical, etc.). See **Notes on extracting** `z` for how `z` is handled when argument `x` is a data structure (rather than a numeric vector).
`rw`	The coordinates of the corners of the rectangular window enclosing the triangulation, in the order (xmin, xmax, ymin, ymax). Any data points outside this window are discarded. If this argument is omitted, it defaults to values given by the range of the data, plus and minus 10 percent.

eps	A value of epsilon used in testing whether a quantity is zero, mainly in the context of whether points are collinear. If anomalous errors arise, it is possible that these may averted by adjusting the value of eps upward or downward.
sort	Logical argument; if TRUE (the default) the data are sorted into a sequence of "bins" prior to triangulation; this makes the algorithm slightly more efficient. Normally one would set sort equal to FALSE only if one wished to observe some of the fine detail of the way in which adding a point to a data set affected the triangulation, and therefore wished to make sure that the point in question was added last. Essentially this argument would get used only in a debugging process.
plot	Logical argument; if TRUE a plot is produced. The nature of the plot may be controlled by using the ... argument to pass appropriate arguments to plot.deldir(). Without "further instruction" a plot of the points being triangulated and of both the triangulation and the tessellation is produced;
round	Logical scalar. Should the data stored in the returned value be rounded to digits decimal digits? This is essentially for cosmetic purposes. This argument is a "new addition" to deldir(), as of version 0.1-26. Previously rounding was done willy-nilly. The change was undertaken when Kodi Arfer pointed out that the rounding might have unwanted effects upon "downstream" operations.
digits	The number of decimal places to which all numeric values in the returned list should be rounded. Defaults to 6. Ignored if round (see above) is set to FALSE.
...	Auxiliary arguments add, wlines, number, nex, col, lty, pch, xlim, and ylim (and possibly other plotting parameters) may be passed to plot.deldir() through ... if plot=TRUE.

Details

This package had its origins (way back in the mists of time!) as an Splus library section named "delaunay". That library section in turn was a re-write of a stand-alone Fortran program written in 1987/88 while the author was with the Division of Mathematics and Statistics, CSIRO, Sydney, Australia. This program was an implementation of the second (iterative) Lee-Schacter algorithm. The stand-alone Fortran program was re-written as an Splus function (which called upon dynamically loaded Fortran code) while the author was visiting the University of Western Australia, May, 1995.

Further revisions were made December 1996. The author gratefully acknowledges the contributions, assistance, and guidance of Mark Berman, of D.M.S., CSIRO, in collaboration with whom this project was originally undertaken. The author also acknowledges much useful advice from Adrian Baddeley, formerly of D.M.S., CSIRO (now Professor of Statistics at Curtin University). Daryl Tingley of the Department of Mathematics and Statistics, University of New Brunswick, provided some helpful insight. Special thanks are extended to Alan Johnson, of the Alaska Fisheries Science Centre, who supplied two data sets which were extremely valuable in tracking down some errors in the code.

Don MacQueen, of Lawrence Livermore National Lab, wrote an Splus driver function for the old stand-alone version of this software. That driver, which was available on Statlib, was deprecated in favour of the Statlib package "delaunay". Don also collaborated in the preparation of the latter package. It is not clear to me whether the "delaunay" package, or indeed Statlib (or indeed Splus!) still exist.

See the ChangeLog for information about further revisions and bug-fixes.

Value

A list (of class deldir), invisible if plot=TRUE, with components:

delsgs — A data frame with 6 columns. The first 4 entries of each row are the coordinates of the points joined by an edge of a Delaunay triangle, in the order (x1,y1,x2,y2). The last two entries are the indices of the two points which are joined.

dirsgs — A data frame with 10 columns. The first 4 entries of each row are the coordinates of the endpoints of one the edges of a Dirichlet tile, in the order (x1,y1,x2,y2). The fifth and sixth entries, in the columns named ind1 and ind2, are the indices of the two points, in the set being triangulated, which are separated by that edge. The seventh and eighth entries, in the columns named bp1 and bp2 are logical values. The entry in column bp1 indicates whether the first endpoint of the corresponding edge of a Dirichlet tile is a boundary point (a point on the boundary of the rectangular window). Likewise for the entry in column bp2 and the second endpoint of the edge.

The ninth and tenth entries, in columns named thirdv1 and thirdv2 are the indices of the respective third vertices of the Delaunay triangles whose circumcentres constitute the corresponding endpoints of the edge under consideration. (The other two vertices of the triangle in question are indexed by the entries of columns ind1 and ind2.)

The entries of columns thirdv1 and thirdv2 may (also) take the values $-1, -2, -3$, and -4. This will be the case if the circumcentre in question lies outside of the rectangular window rw. In these circumstances the corresponding endpoint of the tile edge is the intersection of the line joining the two circumcentres with the boundary of rw, and the numeric value of the entry of column "thirdv1" (respectively "thirdv2") indicates which side. The numbering follows the convention for numbering the sides of a plot region in R: 1 for the bottom side, 2 for the left hand side, 3 for the top side and 4 for the right hand side.

Note that the entry in column thirdv1 will be negative if and only if the corresponding entry in column bp1 is TRUE. Similarly for columns thirdv2 and bp2.

summary — a data frame with 9, 10 or 11 columns and n.data + n.dum rows (see below). The rows correspond to the points in the set being triangulated. Note that the row names are the indices of the points in the original sequence of points being triangulated/tessellated. Usually these will be the sequence 1, 2, ..., n. However if there were *duplicated* points then the row name corresponding to a point is the *first* of the indices of the set of duplicated points in which the given point appears. The columns are:

- x (the x-coordinate of the point)
- y (the y-coordinate of the point)
- z (the auxiliary values or "tags"; present only if these were specified)
- n.tri (the number of Delaunay triangles emanating from the point)

- `del.area` (1/3 of the total area of all the Delaunay triangles emanating from the point)
- `del.wts` (the corresponding entry of the `del.area` column divided by the sum of this column)
- `n.tside` (the number of sides — within the rectangular window — of the Dirichlet tile surrounding the point)
- `nbpt` (the number of points in which the Dirichlet tile intersects the boundary of the rectangular window)
- `dir.area` (the area of the Dirichlet tile surrounding the point)
- `dir.wts` (the corresponding entry of the `dir.area` column divided by the sum of this column).

Note that the factor of 1/3 associated with the del.area column arises because each triangle occurs three times — once for each corner.

`n.data` the number of points in the set which was triangulated, with any duplicate points eliminated. It is the same as the number of rows of `summary`.

`del.area` the area of the convex hull of the set of points being triangulated, as formed by summing the `del.area` column of `summary`.

`dir.area` the area of the rectangular window enclosing the points being triangulated, as formed by summing the `dir.area` column of `summary`.

`rw` the specification of the corners of the rectangular window enclosing the data, in the order (xmin, xmax, ymin, ymax).

`ind.orig` A vector of the indices of the points (x,y) in the set of coordinates initially supplied to `deldir()` before duplicate points (if any) were removed. These indices are used by `triang.list()`.

Side Effects

If `plot=TRUE` a plot of the triangulation and/or tessellation is produced or added to an existing plot.

Notes on extracting the coordinates

The protocol for extracting the x and y coordinates from the arguments `x` and `y` is a bit complicated and confusing. It is designed to handle a number of different desiderata and to accommodate various feature-requests that users have made over the years. Basically the protocol is:

- If `x` is a numeric vector and `y` is a numeric vector then `x` is used as the x-coordinates and `y` is used as the y-coordinates.
- If `x` is a matrix, a data frame, or a generic list), and `y` is a numeric vector, then the x-coordinates are sought amongst the components of `x` and `y` is used as the y-coordinates.
- If `x` is a matrix, a data frame, or a generic list and `y` is not specified or cannot be found, then both the x-coordinates and y-coordinates are sought amongst the components of `x`.
- If `x` an object of class `"ppp"` then both the x-coordinates and y-coordinates are taken from the components of `x`. If `y` is specified, it is ignored (with a warning).
- If `x` is a numeric vector and `y` is not specified or cannot be found, then an error is thrown.

A few more details:

- If x is of class "ppp" then it will definitely have components named "x" and "y".
- If x is a generic list, it *must* have a component named "x" (otherwise an error is thrown), and the x-coordinates are set equal to this component. If y is not specified or cannot be found, then a "y" component of x is sought. If such a component exists then the y-coordinates are set equal to this component. Otherwise an error is thrown).
- If x is a matrix or a data frame, the protocol gets a bit more intricate.
  - If x has a column named "x" then this column is taken to be the x-coordinates.
  - Otherwise the x-coordinates are taken to be the *first* column of x that is not named "y" or znm (where znm is the name of the object providing the "tags", if "tags" have been specified).
  - If there is no such first column (e.g. if there are only two columns and these have names "y" and znm) then an error is thrown.
  - If y is not specified or cannot be found, and if x has a column named "y" then this column is taken to be the y-coordinates.
  - Otherwise, in this situation, the y-coordinates are taken to be the *first* column of x that is not named "x" or znm and is not equal to the column previously selected to be the x-coordinates.
  - If there is no such first column (e.g. if there are only two columns and these have names "x" and znm), then an error is thrown.

Got all that? :-) If these instructions seem rather bewildering (and indeed they are!) just keep things simple and make calls like deldir(x,y) where x and y are numeric vectors that have been previously assigned in the global environment.

Notes on extracting z

If argument x is a data structure (rather than a numeric vector) and is *not* an object of class "ppp" then z, if specified and not found, is searched for in x. If x *is* of class "ppp" then what happens depends on whether z was specified or left to take its default value of NULL. In the former case, z takes the specified value. In the latter case the value of "z" is taken from the marks of x provided that x is indeed a marked point pattern and that the marks are *atomic* (essentially provided that the marks are not a data frame). Otherwise z is left NULL, i.e. there are no "tags" associated with the points.

Notes on "tags"

The "tags" are simply values that are associated in some way with the data points and hence with the tiles of the tessellation produced. They **DO NOT** affect the tessellation. In previous versions of this package (0.2-10 and earlier) the entries of z were referred to as "weights". This terminology has been changed since it is misleading. The tessellation produced when a z argument is supplied is the same as is it would be if there were no z argument (i.e. no "weights"). The deldir package **DOES NOT do weighted tessellation**.

Notes on Memory Allocation

It is difficult-to-impossible to determine *a priori* how much memory needs to be allocated (in the Fortran code) for storing the edges of the Delaunay triangles and Dirichlet tiles, and for storing the "adjacency list" used by the Lee-Schacter algorithm. In the code, an attempt is made to allocate

sufficient storage. If, during the course of running the algorithm, the amount of storage turns out to be inadequate, the algorithm is halted, the storage is incremented, and the algorithm is restarted (with an informative message). This message may be suppressed by wrapping the call to `deldir()` in `suppressMessages()`.

Notes on error messages

In previous versions of this package, error traps were set in the underlying Fortran code for 17 different errors. These were identified by an error number which was passed back up the call stack and finally printed out by `deldir()` which then invisibly returned a `NULL` value. A glossary of the meanings of the values of was provided in a file to be found in a file located in the `inst` directory ("folder" if you are a Windoze weenie).

This was a pretty shaganappi system. Consequently, as of version 1.2-1, conversion to "proper" error trapping was implemented. Such error trapping is effected via the `rexit()` subroutine which is now available in `R`. (See "Writing R Extensions", section 6.2.1.)

Note that when an error is detected, `deldir()` now exits with a genuine error, rather than returning `NULL`. The glossary of the meanings of "error numbers" is now irrelevant and has been removed from the `inst` directory.

An error trap that merits particular mention was introduced in version `0.1-16` of `deldir`. This error trap relates to "triangle problems". It was drawn to my attention by Adam Dadvar (on 18 December, 2018) that in some data sets collinearity problems may cause the "triangle finding" procedure, used by the algorithm to successively add new points to a tessellation, to go into an infinite loop. A symptom of the collinearity is that the vertices of a putative triangle appear *not* to be in anticlockwise order irrespective of whether they are presented in the order `i, j, k` or `k, j, i`. The result of this anomaly is that the procedure keeps alternating between moving to "triangle" `i, j, k` and moving to "triangle" `k, j, i`, forever.

The error trap in question is set in `trifnd`, the triangle finding subroutine. It detects such occurrences of "clockwise in either orientation" vertices. The trap causes the `deldir()` function to throw an error rather than disappearing into a black hole.

When an error of the "triangle problems" nature occurs, a *possible* remedy is to increase the value of the `eps` argument of `deldir()`. (See the **Examples**.) There may conceivably be other problems that lead to infinite loops and so I put in another error trap to detect whether the procedure has inspected more triangles than actually exist, and if so to throw an error.

Note that the strategy of increasing the value of `eps` is *probably* the appropriate response in most (if not all) of the cases where errors of this nature arise. Similarly this strategy is *probably* the appropriate response to the errors

- Vertices of "triangle" are collinear and vertex 2 is not between 1 and 3. Error in circen.
- Vertices of triangle are collinear. Error in dirseg.
- Vertices of triangle are collinear. Error in dirout.

However it is impossible to be sure. The intricacy and numerical delicacy of triangulations is too great for anyone to be able to foresee all the possibilities that could arise.

If there is any doubt as to the appropriateness of the "increase `eps`" strategy, users are advised to do their best to explore the data set, graphically or by other means, and thereby determine what is actually going on and why problems are occurring.

Warnings

1. The process for determining if points are duplicated changed between versions 0.1-9 and 0.1-10. Previously there was an argument `frac` for this function, which defaulted to 0.0001. Points were deemed to be duplicates if the difference in `x`-coordinates was less than `frac` times the width of `rw` and `y`-coordinates was less than `frac` times the height of `rw`. This process has been changed to one which uses `duplicated()` on the data frame whose columns are `x` and `y`.

   As a result it may happen that points which were previously eliminated as duplicates will no longer be eliminated. (And possibly vice-versa.)
2. The components `delsgs` and `summary` of the value returned by `deldir()` are now *data frames* rather than matrices. The component `summary` was changed to allow the "auxiliary" values `z` to be of arbitrary mode (i.e. not necessarily numeric). The component `delsgs` was then changed for consistency. Note that the other "matrix-like" component `dirsgs` has been a data frame since time immemorial.

Acknowledgement

I would like to express my most warm and sincere thanks to Duncan Murdoch (Emeritus Professor of Statistics, Western University) for helping me, with incredible patience and forbearance, to straighten out my thinking in respect of adjustments that I recently (October 2021) made to the argument processing protocol in the `deldir()` function. Duncan provided numerous simple examples to demonstrate when and how things were going wrong, and patiently explained to me how I was getting one aspect of the protocol backwards.

Author(s)

Rolf Turner <r.turner@auckland.ac.nz>

References

Lee, D. T. and Schacter, B. J. (1980). Two algorithms for constructing a Delaunay triangulation, *International Journal of Computer and Information Sciences* **9** (3), pp. 219 – 242.

Ahuja, N. and Schacter, B. J. (1983). Pattern Models. New York: Wiley.

See Also

`plot.deldir()`, `tile.list()`, `triang.list()`

Examples

```
x <- c(2.3,3.0,7.0,1.0,3.0,8.0)
y <- c(2.3,3.0,2.0,5.0,8.0,9.0)

# Let deldir() choose the rectangular window.
dxy1 <- deldir(x,y)

# User chooses the rectangular window.
dxy2 <- deldir(x,y,rw=c(0,10,0,10))

# Put "dummy" points at the corners of the rectangular
```

```
# window, i.e. at (0,0), (10,0), (10,10), and (0,10)
xx <- c(x,0,10,10,0)
yy <- c(y,0,0,10,10)
dxy3 <- deldir(xx,yy,rw=c(0,10,0,10))

# Plot the triangulation created (but not the tessellation).
dxy2 <- deldir(x,y,rw=c(0,10,0,10),plot=TRUE,wl="tr")

# Example of collinearity error.
## Not run:
    dniP <- deldir(niProperties) # Throws an error

## End(Not run)
    dniP <- deldir(niProperties,eps=1e-8) # No error.
# Example of using data stored in a data frame.
dsw  <- deldir(seaweed)
# Example where "data" is of class "ppp".
dtoy <- deldir(toyPattern)
# The "tags", in dtoy$summary$z, are the marks of the toy ppp
# object which consists of the letters "a","b","c" and "d".
#
# Artificial example of tags.
set.seed(42)
trees1  <- sample(c("spruce","birch","poplar","shoe"),20,TRUE)
trees2  <- sample(c("fir","maple","larch","palm"),20,TRUE)
egDat   <- data.frame(x=runif(20),y=runif(20),species=trees1)
tagNm   <- "species"
species <- trees2
dd1     <- deldir(egDat) # No tags.
dd2     <- deldir(egDat,z=species)  # Uses trees1 as the tags.
dd3     <- deldir(egDat,z="species") # Same as dd2.
dd4     <- deldir(egDat,z=tagNm) # Same as dd2 and dd3.
spec    <- species
dd5     <- deldir(egDat,z=spec) # Uses trees2 as the tags.

# Duncan Murdoch's examples.  The deldir() function was not
# handling such examples correctly until Duncan kindly set
# me on the right path.
set.seed(123)
dd6 <- deldir(rnorm(32),rnorm(32),rnorm(32))
#
x <- cbind(x = 1:10, junk = 11:20)
y <- 21:30
z <- 31:40
d7 <- deldir(x=x, y=y, z=z)
#
# print(d7$summary) reveals that x is 1:10, y is 21:30
# and z is 31:40; x[,"junk"] is ignored as it should be.
x <- cbind(x = 1:10, "rnorm(10)" = 11:20)
y <- 21:30
z <- 41:50
d8 <- deldir(x=x, y=y, z=rnorm(10))
#
```

```
# print(d8$summary) reveals that x is 1:10, y is 21:30 and z is a
# vector of standard normal values.  Even though x has a column with
# the name of the z argument i.e. "rnorm(10)" (!!!)  the specified
# value of z takes precedence over this column (and, as per the usual
# R syntax) over the object named "z" in the global environment.
```

`divchain` *Dividing chain.*

Description

Create the "dividing chain" of a Dirichlet tessellation. The tessellation must have been created from a set of points having associated "tags". The dividing chain consists of those edges of Dirichlet tiles which separate points having different values of the given tags.

Usage

```
   divchain(x, ...)
   ## Default S3 method:
divchain(x, y, z, ...)
   ## S3 method for class 'deldir'
divchain(x, ...)
```

Arguments

`x`	Either an object specifying coordinates (in the case of the `"default"` method; see `deldir()` for details) or an object of class "deldir". In the latter case this object must have been created in such a way that the points of the set being tessellated have associate categorical "tags". That is, `deldir()` must have been called with a `z` argument or the `x` argument to `deldir()` must have had an appropriate component which could be taken to be `z`. Note that if the value of `z` that was used was not a factor, it is coerced to one.
`y`	A numeric vector constituting the y-coordinates of the set of points being tessellated. See `deldir()` for details. Not used by the `"deldir"` method.
`z`	A vector or factor specifying "auxiliary" values or "tags". If this argument is left `NULL` then it is extracted, if possible, from the components of `x`. See `deldir()` for further details. If `z` is not a factor it is coerced to one. See `deldir()` for details. Not used by the `"deldir"` method.
`...`	Arguments to be passed to `deldir()`. Not used by the `"deldir"` method.

Value

An object of class "divchain" consisting of a data frame with columns named "x0", "y0", "x1", "y1", "v01", "v02", "v03", "v11", "v12" and "v13".

The columns named "x0" and "y0" consist of the coordinates of one endpoint of an edge of a Dirichlet tile and the columns named "x1" and "y1" consist of the coordinates of the other endpoint.

The columns named "vij", i = 0, 1, j = 1, 2, 3, consist of the indices of the vertices of the Delaunay triangles whose circumcentres constitute the respective endpoints of the corresponding edge of a Dirichlet tile. The entries of column "vi3" may (also) take the values $-1, -2, -3$, and -4. This will be the case if the circumcentre in question lay outside of the rectangular window `rw` (see `deldir()`) enclosing the points being tessellated. In these circumstances the corresponding endpoint of the tile edge is the intersection of the line joining the two circumcentres with the boundary of `rw`, and the numeric value of the entry of column "vi3" indicates which side. The numbering follows the convention for numbering the sides of a plot region in `R`: 1 for the bottom side, 2 for the left side, 3 for the top side and 4 for the right side.

Note that the triple of vertices uniquely identify the endpoint of the tile edge.

The object has an attribute `rw` which is equal to the specification of the rectangular window within which the triangulation/tessellation in question was constructed. (See `deldir()`.)

Note

This function was created in response to a question asked on `stackoverflow.com` by a user named "Dan".

Author(s)

Rolf Turner `<r.turner@auckland.ac.nz>`

See Also

`deldir()` `plot.divchain()`

Examples

```
set.seed(42)
x   <- runif(50)
y   <- runif(50)
z   <- factor(kmeans(cbind(x,y),centers=4)$cluster)
dc1 <- divchain(x,y,z,rw=c(0,1,0,1))
dxy <- deldir(x,y,z=z,rw=c(0,1,0,1))
dc2 <- divchain(dxy)
```

---

duplicatedxy *Determine duplicated points.*

Description

Find which points among a given set are duplicates of others.

Usage

```
duplicatedxy(x, y)
```

Arguments

x	Either a vector of x coordinates of a set of (two dimensional) points, or a list (or data frame) with columns x and y giving the coordinates of a set of such points.
y	A vector of y coordinates of a set of (two dimensional) points. Ignored if x is a list or data frame.

Details

Often it is of interest to associate each Dirichlet tile in a tessellation of a planar point set with the point determining the tile. This becomes problematic if there are *duplicate* points in the set being tessellated/triangulated. Duplicated points are automatically eliminated "internally" by `deldir()`. The association between tiles and the indices of the original set of points is now preserved by the component `ind.orig` of the object returned by `deldir()`. However confusion could still arise.

If it is of interest to associate Dirichlet tiles with the points determining them, then it is better to proceed by eliminating duplicate points to start with. This function (`duplicatedxy()`) provides a convenient way of doing so.

Value

A logical vector of length equal to the (original) number of points being considered, with entries `TRUE` if the corresponding point is a duplicate of a point with a smaller index, and `FALSE` otherwise.

Warning

Which indices will be considered to be indices of duplicated points (i.e. get `TRUE` values) will of course depend on the order in which the points are presented.

Note

The real work is done by the base **R** function `duplicated()`.

Author(s)

Rolf Turner <r.turner@auckland.ac.nz>

See Also

`duplicated()`, `deldir()`

Examples

```
set.seed(42)
xy  <- data.frame(x=runif(20),y=runif(20))
# Lots of duplicated points.
xy  <- rbind(xy,xy[sample(1:20,20,TRUE),])
# Scramble.
ii  <- sample(1:40,40)
x   <- xy$x[ii]
y   <- xy$y[ii]
# Unduplicate!
```

```
iii <- !duplicatedxy(x,y)
xu  <- x[iii]
yu  <- y[iii]
# The i-th tile is determined by (xu[i],yu[i]):
dxy <- deldir(xu,yu)
```

---

grapherXmpl *grapherXmpl*

Description

A data set taken from an example in the grapherator package. This data set demonstrates handling a data set with duplicated points.

Usage

```
grapherXmpl
```

Format

A data frame with 250 observations on the following 2 variables.

`x` a numeric vector

`y` a numeric vector

Details

There are 25 duplicated points, so the net number of observations is 225. These data constitute a structure (named `coordinates`) generated internally in the function `addEdgesDelaunay`. The call is to be found in the examples in the help file for the `plot.grapherator()` in the `grapherator` package. The relevant example initially threw an error, revealing a bug in `deldir()` that was triggered when there were duplicated points in the data.

Source

The `grapherator` package, https://CRAN.R-project.org/package=grapherator

Examples

```
dgX <- deldir(grapherXmpl) # Now works!!!`
```

`lawSummary` *Produce a Lewis-Aboav-Weaire summary of a tessellation.*

Description

Produce a summary of a Dirichlet (Voronoi) tessellation in terms of parameters relevant to Lewis's law and Aboav-Weaire's law. Note that "law" in the function name corresponds to "Lewis-Aboav-Weaire".

Usage

```
lawSummary(object)
```

Arguments

`object`	An object of class `"deldir"` as returned by the function `deldir()`.

Details

Tiles are stripped away from the tessellation in "layers". Layer 1 consists of "boundary" tiles, i.e. tiles having at least one vertex on the enclosing rectangle (determined by the `rw` argument of `deldir()`). Layer 2 consists of tiles which are neighbours of tiles in layer 1 (i.e. tiles determined by points that are Delaunay neighbours of points determining the tiles in layer 1). Layer 3 consists of tiles which are neighbours of tiles in layer 2.

The parameters of interest in respect of the Lewis-Aboav-Weaire summary are then calculated in terms of the tiles that remain after the three layers have been stripped away, which will be referred to as "interior" tiles. These parameters are:

- the areas of each of the interior tiles
- the number of edges of each of the interior tiles
- the number of edges of all neighbouring tiles of each of the interior tiles.

Note that the neighbouring tiles of the interior tiles may include tiles which are *not themselves* interior tiles (i.e. tiles which are in layer 3).

This function was created at the request of Kai Xu (Fisheries College, Jimei University, Xiamen, Fujian, China 361021).

Value

If no tiles remain after the three layers have been stripped away, then the returned value is `NULL`. Otherwise the returned value is a list with components calculated in terms of the remaining ("interior") tiles. These components are:

- `tile.vertices` A list whose entries are data frames giving the coordinates of the vertices of the interior tiles.
- `tile.areas` A vector of the areas of the interior tiles in the tessellation in question.

- tile.tags A vector or factor whose values are the "tags" of the interior tiles. The "original" of this object (the "tags" associated with all of the tiles) is provided as the z argument to deldir(). The tile.tags component of the value returned by lawSummary() is present only if deldir() was called with a (non-NULL) value of the z argument.
- num.edges A vector of the number of edges of each such tile.
- num.nbr.edges A list with a component for each point, in the set being tessellated, whose corresponding tile is an interior tile. Each component of this list is the vector of the number of edges of the interior tiles determined by points which are Delaunay neighbours of the point corresponding to the list component in question.
- totnum.nbr.edges A vector whose entries consist of the sums of the vectors in the foregoing list.

The returned list also has attributes as follows:

- i1 An integer vector whose entries are in the indices of the tiles in layer 1.
- i2 An integer vector whose entries are in the indices of the tiles in layer 2.
- i3 An integer vector whose entries are in the indices of the tiles in layer 3.
- i.kept An integer vector whose entries are in the indices of the tiles that are kept, i.e. those that remain after the three layers have been stripped away.

Author(s)

Rolf Turner <r.turner@auckland.ac.nz>

See Also

tile.list() tile.centroids()

Examples

```
# A random pattern:
set.seed(42)
xy1   <- data.frame(x=runif(400,0,20),y=runif(400,0,20))
dxy1  <- deldir(xy1)
ldxy1 <- lawSummary(dxy1)
tl1   <- tile.list(dxy1)
plot(0,0,type="n",xlim=c(-2,35),ylim=c(0,20),asp=1,xlab="x",ylab="y",bty="l")
plot(tl1,showpoints=FALSE,add=TRUE)
points(xy1[attr(ldxy1,"i1"),],pch=20,col="yellow")
points(xy1[attr(ldxy1,"i2"),],pch=20,col="blue")
points(xy1[attr(ldxy1,"i3"),],pch=20,col="green")
points(xy1[attr(ldxy1,"i.kept"),],pch=20,col="red")
legend("right",pch=20,col=c("yellow","blue","green","red"),
       legend=c("layer 1","layer 2","layer 3","interior"))

# A highly structured pattern (example due to Kai Xu):
set.seed(115)
x  <- c(rep(1:20,10),rep((1:20)+0.5,10))
y  <- c(rep(1:10,each=20),rep((1:10)+0.5,each=20))*sqrt(3)
a  <- runif(400,0,2*pi)
```

```
b  <- runif(400,-1,1)
x  <- x+0.1*cos(a)*b
y  <- y+0.1*sin(a)*b
xy2 <- data.frame(x,y)
dxy2  <- deldir(xy2)
ldxy2 <- lawSummary(dxy2)
tl2   <- tile.list(dxy2)
plot(0,0,type="n",xlim=c(-2,35),ylim=c(0,20),asp=1,xlab="x",ylab="y",bty="l")
plot(tl2,showpoints=FALSE,add=TRUE)
points(xy2[attr(ldxy2,"i1"),],pch=20,col="yellow")
points(xy2[attr(ldxy2,"i2"),],pch=20,col="blue")
points(xy2[attr(ldxy2,"i3"),],pch=20,col="green")
points(xy2[attr(ldxy2,"i.kept"),],pch=20,col="red")
legend("right",pch=20,col=c("yellow","blue","green","red"),
       legend=c("layer 1","layer 2","layer 3","interior"))
```

`niProperties` *Northern Ireland properties.*

Description

The locations (in longitude and latitude) of a number of properties (land holdings) in Northern Ireland.

Usage

```
data("niProperties")
```

Format

A data frame with 240 observations on the following 2 variables.

`x` A numeric vector of longitudes.

`y` A numeric vector of latitudes.

Source

These data were kindly provided by Adam Dadvar of the *Cartesian Limited* consulting service. URL: `http://www.cartesian.com`.

Examples

```
# data(niProperties)
# It is unnecessary to use \code{data} since \code{niProperties} is
# a "first class object".  It is "lazily loaded".
plot(niProperties)
```

plot.deldir *Plot objects produced by deldir*

Description

This is a method for plot.

Usage

```
## S3 method for class 'deldir'
plot(x,add=FALSE,wlines=c("both","triang","tess"),
                      showpoints=TRUE,number=FALSE,cex=1,nex=1,
                      cmpnt_col=NULL,cmpnt_lty=NULL,pch=1,xlim=NULL,
                      ylim=NULL,axes=FALSE,xlab=if(axes) "x" else "",
                      ylab=if(axes) "y" else"",showrect=FALSE,asp=1,...)
```

Arguments

x	An object of class "deldir" as returned by the function deldir.
add	logical argument; should the plot be added to an existing plot?
wlines	"which lines?". I.e. should the Delaunay triangulation be plotted (wlines="triang"), should the Dirichlet tessellation be plotted (wlines="tess"), or should both be plotted (wlines="both", the default) ?
showpoints	Logical scalar; should the points being triangulated/tessellated be plotted?
number	Logical argument, defaulting to FALSE; if TRUE then the points plotted will be labelled with their index numbers (corresponding to the row numbers of the matrix "summary" in the output of deldir).
cex	The value of the character expansion argument cex to be used with the plotting symbols for plotting the points.
nex	The value of the character expansion argument cex to be used by the text function when numbering the points with their indices. Used only if number=TRUE.
cmpnt_col	A vector or list specifying the colours to be used for plotting the (up to five) different components of the graphic, explicitly the triangulation, the tessellation, the data points, the point numbers and the enclosing rectangle (x$rw) in that order. The components of this vector or list may be named, with the names chosen from "tri","tess","points","num","rect". The default is c(tri=1,tess=1,points=1,num=1,rect=1). If the vector or list is not named, the component names are assumed to be from the default list of names in the given order. Colours may be specified as positive integers, corresponding to the entries of the current palette(), or as names of colours (from the list given by colors()), or in terms of their RGB components with strings of the form "#RRGGBB". (See **Color Specification** in par().) If fewer than five colours are given, the missing components are filled in with 1 or palette()[1] as is appropriate. If cmpnt_col does not have names then

the components are simply recycled. If more than five colours are given, the redundant ones are ignored (i.e. only the first five are used).

`cmpnt_lty` A vector or list (of length two) of line types for plotting the two components of the graphic that consist of lines, i.e. the triangulation and the tessellation. The types may consist of integers between 1 and 6, or of approriate text strings ("solid", "dashed", "dotted", "dotdash", "longdash" or "twodash"; see the discussion of `lty` in `par()`). The components of `cmpnt_lty` (vector or list) may have names (`"tri","tess"`). The default is c(tri=1,tess=2). If a single value is given, the missing one is filled in as `1` or `"solid"` as is appropriate. If more than two values are given, the redundant ones are ignored, i.e. only the first two are used.

`pch` The plotting symbol for plotting the points. May be either integer or character.

`xlim` The limits on the x-axis. Defaults to `rw[1:2]` where `rw` is the rectangular window specification returned by deldir().

`ylim` The limits on the y-axis. Defaults to `rw[3:4]` where `rw` is the rectangular window specification returned by deldir().

`axes` Logical scalar. Should axes be drawn on the plot?

`xlab` Label for the x-axis. Defaults to `x` if `axes` is `TRUE` and to the empty string if `axes` is `FALSE`. Ignored if `add=TRUE`.

`ylab` Label for the y-axis. Defaults to `y` if `axes` is `TRUE` and to the empty string if `axes` is `FALSE`. Ignored if `add=TRUE`.

`showrect` Logical scalar; show the enclosing rectangle `rw` (see `deldir()`) be plotted?

`asp` The aspect ratio of the plot; integer scalar or `NA`. Set this argument equal to `NA` to allow the data to determine the aspect ratio and hence to make the plot occupy the complete plotting region in both `x` and `y` directions. This is inadvisable; see the **Warning**. The `asp` argument is ignored if `add` is `TRUE`.

`...` Further plotting parameters (e.g. `lw`, `col`) to be passed to `segments()`. (Conceivably one might also use `...` to supply additional arguments to be passed to `points()` or `text()` but this is unlikely.) Note that if `col` is specified as one of the `...` arguments then this is used as the `col` argument of `segments` (and the values of `cmpnt_col[1:2]` are ignored.

Side Effects

A plot of the edges of the Delaunay triangles and/or of the Dirichlet tiles is produced or added to an existing plot. By default the triangles are plotted with solid lines (lty=1) and the tiles with dotted lines (lty=2). In addition the points being triangulated may be plotted.

Warning

In previous versions of the `deldir` package, the aspect ratio was not set. Instead, the shape of the plotting region was set to be square (`pty="s"`). This practice was suboptimal. To reproduce previous behaviour set `asp=NA` (and possibly `pty="s"`) in the call to `plot.deldir()`. Users, unless they *really* understand what they are doing and why they are doing it, are now *strongly advised* not to set the value of `asp` but rather to leave `asp` equal to its default value of `1`. Any other value may distort the tessellation and destroy the perpendicular appearance of lines which are indeed

perpendicular. (And conversely may cause lines which are not perpendicular to appear as if they are.)

Author(s)

Rolf Turner <r.turner@auckland.ac.nz>

See Also

deldir()

Examples

```
x <- c(2.3,3.0,7.0,1.0,3.0,8.0) + 0.5
y <- c(2.3,3.0,2.0,5.0,8.0,9.0) + 0.5
x <- c(x,1,10,10,1)
y <- c(y,1,1,10,10)
dxy <- deldir(x,y,rw=c(0,11,0,11))
plot(dxy)

# Plots the tessellation, but does not save the results:
deldir(x,y,rw=c(0,11,0,11),plot=TRUE,
       wl="tess",cmpnt_col=c(1,1,2,3,4),num=TRUE)

# Plots the triangulation, but not the tessellation or the points,
# and saves the returned structure:

dxy <- deldir(x,y,rw=c(0,11,0,11),plot=TRUE,wl="triang",wp="none")

# Plot everything:
plot(dxy,cmpnt_col=c("orange","green","red","blue","black"),cmpnt_lty=1,
     number=TRUE,nex=1.5,pch=c(19,9),showrect=TRUE,axes=TRUE)

# Complicated example from He Huang:
# "Linguistic distances".
vldm <- c(308.298557,592.555483,284.256926,141.421356,449.719913,
          733.976839,591.141269,282.842712,1.414214,732.562625)
ldm <- matrix(nrow=5,ncol=5)
ldm[row(ldm) > col(ldm)]  <- vldm
ldm[row(ldm) <= col(ldm)] <- 0
ldm <- (ldm + t(ldm))/2
rownames(ldm) <- LETTERS[1:5]
colnames(ldm) <- LETTERS[1:5]

# Data to be triangulated.
id <- c("A","B","C","D","E")
x  <- c(0.5,1,1,1.5,2)
y  <- c(5.5,3,7,6.5,5)
dat_Huang <- data.frame(id = id, x = x, y = y)

# Form the triangulation/tessellation.
dH <- deldir(dat_Huang)
```

```
# Plot the triangulation with line widths proportional
# to "linguistic distances".
all_col <- rainbow(100)
i       <- pmax(1,round(100*vldm/max(vldm)))
use_col <- all_col[i]
ind     <- as.matrix(dH$delsgs[,c("ind1","ind2")])
lwv <- ldm[ind]
lwv <- 10*lwv/max(lwv)
plot(dH,wlines="triang",col=use_col,showpoints=FALSE,
     lw=lwv,asp=NA)
with(dH,text(x,y,id,cex=1.5))
```

`plot.divchain` *Plot a dividing chain.*

Description

Plot the dividing chain of a Dirichlet tessellation. The tessellation must have been created from a set of points having associated categorical "tags". The dividing chain consists of those edges of Dirichlet tiles which separate points having different values of the given tags.

Usage

```
   ## S3 method for class 'divchain'
plot(x, add = FALSE, ...)
```

Arguments

`x`	An object of class "divchain". See `divchain.deldir()` for details.
`add`	Logical scalar. It `add=TRUE` the plot of the dividing chain is added to an existing plot.
`...`	Graphical parameters such as `main`, `xlab`, `col.main`, `col.lab`. In particular if `bty` is supplied (as a value other than `n`) a "box" will be drawn around the plot that is formed when `add=FALSE`. Also a non-standard graphical parameter `boxcol` may be supplied which will be taken to be the colour with which the box is drawn. If a `col` argument is supplied, this determines the colour for plotting the segments constituting the dividing chain.

Value

None.

Note

This function was created in response to a question asked on `stackoverflow.com` by a user named "Dan".

Author(s)

Rolf Turner <r.turner@auckland.ac.nz>

See Also

divchain() divchain.default() divchain.deldir() deldir()

Examples

```
set.seed(42)
x   <- runif(50)
y   <- runif(50)
z   <- factor(kmeans(cbind(x,y),centers=4)$cluster)
dc  <- divchain(x,y,z,rw=c(0,1,0,1))
plot(dc,lwd=2,col="blue",bty="o")
```

---

plot.tile.list *Plot Dirichlet (Voronoi) tiles*

Description

A method for plot. Plots (sequentially) the tiles associated with each point in the set being tessellated.

Usage

```
## S3 method for class 'tile.list'
plot(x, verbose = FALSE, close = FALSE, pch = 1,
                         fillcol = getCol(x,warn=warn), col.pts=NULL,
                         col.num=NULL,border=NULL, showpoints = !number,
                         add = FALSE, asp = 1, clipp=NULL, xlab = "x",
                         ylab = "y", main = "", warn=TRUE,
                         number=FALSE,adj=NULL,...)
```

Arguments

x	A list of the tiles in a tessellation, as produced the function tile.list().
verbose	Logical scalar; if TRUE the tiles are plotted one at a time (with a "Go?" prompt after each) so that the process can be watched.
close	Logical scalar; if TRUE the outer edges of of the tiles (i.e. the edges which are contained in the enclosing rectangle) are drawn. Otherwise tiles on the periphery of the tessellation are left "open".
pch	The plotting character (or vector of plotting characters) with which to plot the points of the pattern which was tessellated. Ignored if showpoints is FALSE.

`fillcol` Optional vector (possibly of length 1, i.e. a scalar) whose entries can be interpreted as colours by `col2rgb()`. The i-th entry indicates with which colour to fill the i-th tile. Note that an `NA` entry cause the tile to be left unfilled. This argument will be replicated to have length equal to the number of tiles. The default value is created (using the tile "tags", if these exist) by the undocumented function `getCol()`.

`col.pts` Optional vector like unto `fillcol` whose entries can be interpreted as colours by `col2rgb()`. The i-th entry indicates with which colour to plot the i-th point. This argument will be replicated to have length equal to the number of tiles. Ignored if `showpoints` is `FALSE`.

`col.num` Optional vector like unto `col.pts`. Determines the colours in which the point numbers (see `number` below) are plotted. This argument will be replicated to have length equal to the number of tiles. Ignored if `number` is `FALSE`.

`border` A scalar that can be interpreted as a colour by `col2rgb()`, indicating the colour with which to plot the tile boundaries. Defaults to black unless all of the fill colours specified by `fillcol` are black, in which case it defaults to white. If `length(border) > 1` then only its first entry is used.

`showpoints` Logical scalar; if `TRUE` the points of the pattern which was triangulated/tessellated are plotted.

`add` Logical scalar; should the plot of the tiles be added to an existing plot?

`asp` The aspect ratio of the plot; integer scalar or `NA`. Set this argument equal to `NA` to allow the data to determine the aspect ratio and hence to make the plot occupy the complete plotting region in both `x` and `y` directions. This is inadvisable; see the **Warnings**.

`clipp` An object specifying a polygon to which the tessellation being plotted should be clipped. It should consist either of:

- a list containing two components `x` and `y` giving the coordinates of the vertices of a single polygon. The last vertex should not repeat the first vertex. Or:
- a list of list(x,y) structures giving the coordinates of the vertices of several polygons.

If this argument is provided then the plot of the tessellation is "clipped" to the polygon specified by `clipp`.

`xlab` Label for the `x`-axis (used only if `add` is `FALSE`).

`ylab` Label for the `y`-axis (used only if `add` is `FALSE`).

`main` A title for the plot (used only if `add` is `FALSE`).

`warn` Logical scalar passed to the internal function `getCol()`. Should a warning be issued if the `z` components of the entries of `x` cannot all be interpreted as colours. See **Notes**.

`number` Logical scalar; if `TRUE` the numbers of the points determining the tiles are plotted in the tiles. Note that if `number` is `TRUE` then `showpoints` defaults to `FALSE`

`adj` The "adjustment" argument to `text()`. If `number` and `showpoints` are both `TRUE` it defaults to `-1` (so that the numbers and point symbols are not superimposed). If `number` is `TRUE` and `showpoints` is `FALSE` it defaults to `0`. If `number` is `FALSE` it is ignored.

`...` Optional arguments; may be passed to `points()` and `text()`.

Value

NULL; side effect is a plot.

Warnings

- The behaviour of this function with respect to "clipping" has changed substantially since the previous release of `deldir`, i.e. 1.1-0. The argument `clipwin` has been re-named `clipp` ("p" for "polygon"). Clipping is now effected via the new package `polyclip`. The `spatstat` package is no longer used. The argument `use.gpclib` has been eliminated, since `gpclib` (which used to be called upon by `spatstat` has been superseded by `polyclip` which has an unrestrictive license.
- As of release 0.1-1 of the `deldir` package, the argument `fillcol` to this function *replaces* the old argument `polycol`, but behaves somewhat differently.
- The argument `showrect` which was present in versions of this function prior to release 0.1-1 has been eliminated. It was redundant.
- As of release 0.1-1 the `col.pts` argument *might* behave somewhat differently from how it behaved in the past.
- The arguments `border`, `clipp`, and `warn` are new as of release 0.1-1.
- Users, unless they *really* understand what they are doing and why they are doing it, are *strongly advised* not to set the value of `asp` but rather to leave `asp` equal to its default value of `1`. Any other value distorts the tesselation and destroys the perpendicular appearance of lines which are indeed perpendicular. (And conversely can cause lines which are not perpendicular to appear as if they are.)

Notes

- If `clipp` is not `NULL` and `showpoints` is `TRUE` then it is possible that some of the points "shown" will not fall inside any of the plotted tiles. (This will happen if the parts of the tiles in which they fall have been "clipped" out.) If a tile is clipped out *completely* then the point which determines that tile is *not* plotted irrespective of the value of `showpoints`.
- If the `z` components of the entries of `x` cannot all be interpreted as colours (e.g. if there *aren't* any `z` components, which will be the case if no such values were supplied in the call to `deldir()`) then the internal function `getCol()` returns `NA`. This value of `fillcol` results (as is indicated by the argument list entry for `fillcol`) in (all of) the tiles being left unfilled.
- The new behaviour in respect of the colours with which to fill the plotted tiles, and the argument `clipp` were added at the request of Chris Triggs.
- The argument `asp` was added at the request of Zubin Dowlaty (who presumably knows what he's doing!).

Author(s)

Rolf Turner <r.turner@auckland.ac.nz>

See Also

`deldir()`, `tile.list()`, `triang.list()`, `plot.triang.list()`

Examples

```
set.seed(42)
x <- runif(20)
y <- runif(20)
z <- deldir(x,y,rw=c(0,1,0,1))
w <- tile.list(z)
plot(w)
ccc <- heat.colors(20) # Or topo.colors(20), or terrain.colors(20)
                       # or cm.colors(20), or rainbow(20).
plot(w,fillcol=ccc,close=TRUE)
if(require(polyclip)) {
   CP  <- list(x=c(0.49,0.35,0.15,0.20,0.35,0.42,
                   0.43,0.62,0.46,0.63,0.82,0.79),
               y=c(0.78,0.86,0.79,0.54,0.58,0.70,
                   0.51,0.46,0.31,0.20,0.37,0.54))
   cul <- rainbow(10)[c(1,7,3:6,2,8:10)] # Rearranging colours to improve
                                         # the contrast between contiguous tiles.
   plot(w,clipp=CP,showpoints=FALSE,fillcol=cul)
}
plot(w,number=TRUE,col.num="red")
plot(w,number=TRUE,col.num="red",cex=0.5)
plot(w,showpoints=TRUE,number=TRUE,col.pts="green",col.num="red")
```

`plot.triang.list` *Plot Delaunay triangles*

Description

A method for `plot`. Plots the triangles of a Delaunay triangulation of a set of points in the plane.

Usage

```
## S3 method for class 'triang.list'
plot(x, showrect = FALSE, add = FALSE,
             xlab = "x", ylab = "y", main = "", asp = 1,
             rectcol="black", ...)
```

Arguments

`x`	An object of class "triang.list" as produced by `triang.list()`.
`showrect`	Logical scalar; show the enclosing rectangle `rw` (see `deldir()`) be plotted?
`add`	Logical scalar; should the plot of the triangles be added to an existing plot?
`xlab`	Label for the x-axis.
`ylab`	Label for the y-axis.
`main`	A title for the plot (used only if add is `FALSE`).

asp	The aspect ratio of the plot; integer scalar or NA. Set this argument equal to NA to allow the data to determine the aspect ratio and hence to make the plot occupy the complete plotting region in both x and y directions. This is inadvisable; see the **Warnings**.
rectcol	Text string or integer specifying the colour in which the enclosing rectangle should be plotted. Ignored unless showrect is TRUE.
...	Arguments passed to polygon() which does the actual plotting of the triangles.

Value

None. This function has the side effect of producing (or adding to) a plot.

Warnings

Users are *strongly advised* not to set the value of asp (unless they really know what they are doing) but rather to leave asp equal to its default value of 1. Any other value distorts the tesselation and destroys the perpendicular appearance of lines which are indeed perpendicular. (And conversely can cause lines which are not perpendicular to appear as if they are.)

The argument asp was added at the request of Zubin Dowlaty (who presumably knows what he is doing!).

Author(s)

Rolf Turner <r.turner@auckland.ac.nz>

See Also

deldir(), plot.triang.list(), tile.list(), plot.tile.list()

Examples

```
set.seed(42)
x <- runif(20)
y <- runif(20)
d <- deldir(x,y)
ttt <- triang.list(d)
plot(ttt,border="red",showrect=TRUE,rectcol="green")
sss <- tile.list(d)
plot(sss)
plot(ttt,add=TRUE,border="blue",showrect=TRUE,rectcol="red")
```

print.deldir *Print some information about a tessellation/triangulation.*

Description

Prints a very brief description of an object of class "deldir" as returned by deldir().

Usage

```
## S3 method for class 'deldir'
print(x,digits=NULL,...)
```

Arguments

x	A Delaunay triangulation and Dirichlet (Voronoi) tessellation of a set of points (object of class "deldir").
digits	Integer scalar. The number of digits to which to round the numeric information before printing. Note this may be give negative values. (See round().)
...	Not used.

Details

This is a method for the generic print() function.

Author(s)

Rolf Turner <r.turner@auckland.ac.nz>

See Also

print()

Examples

```
set.seed(42)
x    <- rnorm(200,0,4)
y    <- rnorm(200,0,4)
dxy1 <- deldir(x,y)
dxy2 <- deldir(x,y,rw=c(-12,12,-11,11))
dxy1
dxy2
print(dxy1,digits=4)
```

print.tileInfo *Print a summary of tile information.*

Description

Print a reasonably readable summary of an object of class `tileInfo` as produced by the `tileInfo()` function.

Usage

```
## S3 method for class 'tileInfo'
print(x, digits = 4, ...)
```

Arguments

`x`	An object of class `tileInfo` as produced by the `tileInfo()` function.
`digits`	The (maximum) number of decimal digits to which the output is to be printed.
`...`	Not used. Present for compatibility with the generic `print()` function.

Details

The list produced by `tileInfo()` is a bit messy and hard to comprehend, especially if there is a large number of tiles. This print method produces a screen display which is somewhat more perspicuous.

There are four components to the display:

- A matrix, each row of which is the vector of edge lengths of the tile. The number of columns is the *maximum* of the lengths of the edge length vectors. Rows corresponding to shorter vectors are filled in with blanks. The row names of the matrix indicate the number of the point corresponding to the tile. Note that this number is the index of the point in the original sequence of points that is being tessellated.
- A table of the edge counts of the tiles.
- A simple print out of the areas of the tiles (rounded to a maximum of `digits` decimal digits).
- A simple print out of the perimeters of the tiles (rounded to a maximum of `digits` decimal digits).

This screen display is for "looking at" only. In order to do further calculations on the output of `tileInfo` it is necessary to delve into the bowels of `x` and extract the relevant bits.

In order to get a decent looking display you may (if there are tiles with a large number of edges) need to widen the window in which you are displaying the output and increase the value of the `width` option. E.g. use `options(width=120)`.

Value

None.

Author(s)

Rolf Turner <r.turner@auckland.ac.nz>

See Also

tileInfo()

Examples

```
set.seed(179)
x    <- runif(100)
y    <- runif(100)
dxy  <- deldir(x,y,rw=c(0,1,0,1))
ixy1 <- tileInfo(dxy)
print(ixy1)
ixy2 <- tileInfo(dxy,bndry=TRUE)
print(ixy2)
if(require(polyclip)) {
    CP <- list(x=c(0.49,0.35,0.15,0.20,0.35,0.42,
                   0.43,0.62,0.46,0.63,0.82,0.79),
               y=c(0.78,0.86,0.79,0.54,0.58,0.70,
                   0.51,0.46,0.31,0.20,0.37,0.54))
    ixy3 <- tileInfo(dxy,clipp=CP)
    options(width=120) # And enlarge the console window.
    print(ixy3) # 33 tiles are retained.
    print(ixy3$perimeters$perComps) # The tiles for points 9 and 94 have
                                    # been split into two components.
}
```

---

seaweed *seaweed*

Description

A data frame whose columns are the coordinates of the centroids of the cells in a seaweed frond. The points are estimates-by-eye of where the centroids of the cells occur.

Usage

```
data("seaweed")
```

Format

A data frame with 266 observations on the following 2 variables.

- x The x-coordinates of the cell centroids.
- y The y-coordinates of the cell centroids.

Source

These data were kindly supplied by Dr. John Bothwell of the Department of Biosciences, Durham University. The data were collected by Kevin Yun and Georgia Campbell, members of Dr. Bothwell's research group.

Examples

```
# data(seaweed)
# It is unnecessary to use \code{data} since \code{seaweed} is
# a "first class object".  It is "lazily loaded".

dsw <- deldir(seaweed)
isw <- tileInfo(dsw)
# Expand the width of the terminal window.
options(width=120)
isw
tsw <- tile.list(dsw)
plot(tsw,number=TRUE,col.num="red",cex=0.5,adj=0.5)
```

---

tile.centroids *Compute centroids of Dirichlet (Voronoi) tiles*

Description

Given a list of Dirichlet tiles, as produced by `tile.list()`, produces a data frame consisting of the centroids of those tiles.

Usage

```
tile.centroids(tl)
```

Arguments

`tl`	A list of the tiles (produced by `tile.list()`) in a Dirichlet tessellation of a set of planar points.

Value

A data frame with two columns named `x` and `y`. Each row of this data frame constitutes the centroid of one of the Dirichlet tiles.

Author(s)

Rolf Turner <r.turner@auckland.ac.nz>

References

URL http://en.wikipedia.org/wiki/Centroid

See Also

tile.list()

Examples

```
set.seed(42)
x <- runif(20)
y <- runif(20)
d <- deldir(x,y)
l <- tile.list(d)
g <- tile.centroids(l)
plot(l,close=TRUE)
points(g,pch=20,col="red")
```

---

tile.list *Create a list of tiles in a tessellation*

Description

For each point in the set being tessellated produces a list entry describing the Dirichlet/Voronoi tile containing that point.

Usage

```
tile.list(object,minEdgeLength=NULL,clipp=NULL)
```

Arguments

object	An object of class deldir as produced by the function deldir().
minEdgeLength	Positive numeric scalar specifying the minimum length that an edge of a tile may have. It is used to eliminate edges that are effectively of zero length, which can cause tiles to be "invalid". This argument defaults to sqrt(.Machine$double.eps) time the diameter (length of the diagonal) of the "rectangular window" associated with the tessellation. This rectangular window is available as the rw component of object.
clipp	An object specifying a polygon to which the tessellation, whose tiles are being determined, should be clipped. It should consist either of: • a list containing two components x and y giving the coordinates of the vertices of a single polygon. The last vertex should not repeat the first vertex. Or: • a list of list(x,y) structures giving the coordinates of the vertices of several polygons. If this argument is provided then the tiles in the list that is produced are "clipped" to the polygon specified by clipp. Empty tiles (those which do not intersect the polygon specified by clipp) are omitted. The clipping process may subdivide some of the tiles into two or more discontiguous parts.

Value

A list with one entry for each of the points in the set being tessellated, or for each of the tiles that are retained after clipping if `clipp` is not `NULL`. Each entry is in turn a list with a number of components. These components always include:

`ptNum`	The index of the point in the original sequence of points that is being tessellated. Note that if a point is one of a set of *duplicated* points then `ptNum` is the *first* of the indices of the points in this set.
`pt`	The coordinates of the point whose tile is being described.
`area`	The area of the tile.

If the tile in question has *not* been subdivided by the clipping process then the list components also include:

`x`	The `x` coordinates of the vertices of the tile, in anticlockwise order.
`y`	The `y` coordinates of the vertices of the tile, in anticlockwise order.
`bp`	Vector of logicals indicating whether the tile vertex is a "real" vertex, or a *boundary point*, i.e. a point where the tile edge intersects the boundary of the enclosing rectangle.

If the tile in question *has* been subdivided then the list does not have the foregoing three components but rather has a component `tileParts` which is in turn a list of length equal to the number of parts into which the tile was subdivided. Each component of `tileParts` is yet another list with four components `x`, `y`, `bp` and `area` as described above and as are appropriate for the part in question.

`z`	The "auxiliary value" or "tag" associated with the `pt`; present only if such values were supplied in the call to `deldir()`.

Acknowledgement

The author expresses sincere thanks to Majid Yazdani who found and pointed out a serious bug in `tile.list` in a previous version (0.0-5) of the `deldir` package.

Warning

The set of vertices of each tile may be "incomplete". Only vertices which lie within the enclosing rectangle, and "boundary points" are listed.

Note that the enclosing rectangle may be specified by the user in the call to `deldir()`.

In contrast to some earlier versions of `deldir`, the corners of the enclosing rectangle are now included as vertices of tiles. I.e. a tile which in fact extends beyond the rectangular window and contains a corner of that window will have that corner added to its list of vertices. Thus the corresponding polygon is the intersection of the tile with the enclosing rectangle.

Author(s)

Rolf Turner <r.turner@auckland.ac.nz>

See Also

deldir(), plot.tile.list() triang.list() plot.triang.list()

Examples

```
set.seed(42)
x <- runif(20)
y <- runif(20)
z <- deldir(x,y)
w <- tile.list(z)

z <- deldir(x,y,rw=c(0,1,0,1))
w <- tile.list(z)

z <- deldir(x,y,rw=c(0,1,0,1),dpl=list(ndx=2,ndy=2))
w <- tile.list(z)
if(require(polyclip)) {
    CP <- list(x=c(0.49,0.35,0.15,0.20,0.35,0.42,
                   0.43,0.62,0.46,0.63,0.82,0.79),
               y=c(0.78,0.86,0.79,0.54,0.58,0.70,
                   0.51,0.46,0.31,0.20,0.37,0.54))
    wc <- tile.list(z,clipp=CP) # 10 tiles are retained; the third tile,
                                # corresponding to point 6, is
                                # subdivided into two parts.
}
```

---

tileArea *Area of a Dirichlet tile.*

Description

Calculates the area of a Dirichlet tile, applying a discrete version of Stoke's theorem.

Usage

```
tileArea(x, y, rw)
```

Arguments

x	The x-coordinates of the vertices of the tile, in **anticlockwise** direction. The last coordinate should **not** repeat the first.
y	The y-coordinates of the vertices of the tile, in **anticlockwise** direction. The last coordinate should **not** repeat the first.
rw	A vector of length 4 specifying the rectangular window in which the relevant tessellation was constructed. See deldir() for more detail. Actually this can be any rectangle containing the tile in question.

Details

The heavy lifting is done by the Fortran subroutine `stoke()` which is called by the `.Fortran()` function.

Value

A positive scalar.

Author(s)

Rolf Turner <r.turner@auckland.ac.nz>

See Also

`deldir()` `tilePerim()`

Examples

```
set.seed(42)
x <- runif(20)
y <- runif(20)
z <- deldir(x,y,rw=c(0,1,0,1))
w <- tile.list(z)
with(w[[1]],tileArea(x,y,rw=z$rw))
sapply(w,function(x,rw){tileArea(x$x,x$y,attr(w,"rw"))})
x <- c(0.613102,0.429294,0.386023,0.271880,0.387249,0.455900,0.486101)
y <- c(0.531978,0.609665,0.597780,0.421738,0.270596,0.262953,0.271532)
# The vertices of the Dirichlet tile for point 6.
tileArea(x,y,rw=c(0,1,0,1))
tileArea(x,y,rw=c(-1,2,-3,4)) # Same as above.
```

---

tileInfo *Extract information from a tile list.*

Description

Produces a summary of information about the tiles in an object of class `deldir` as produced by the function `deldir()`.

Usage

```
tileInfo(object, bndry = FALSE, clipp=NULL)
```

Arguments

object	An object of class deldir as produced by the function deldir().
bndry	Logical scalar. If TRUE then the "boundary" tiles (those tiles having edges forming part of the "rectangular window" enclosing the tessellation) are included in the summary. Otherwise they are not included.
clipp	An object specifying a polygon to which the tiles of the tessellation should be clipped. See link{tile.list}() for more information.

Value

An object of class "tileInfo" which consists of a list with components:

indivTiles	This is itself a list. If clipp is NULL then this list has one entry for each tile in "object". If clipp is not NULL then tiles are retained only if they have non-empty intersection with the polygon specified by clipp. The list indivTiles is in fact a *named* list, the names being of form pt.n, where n is equal to the value of ptNum (see below) corresponding to the tile. The entries of indivTiles are themselves in turn lists with entries • edges: a matrix whose rows consists of the x and y coordinates of the endpoints of each edge of the tile • edgeLengths: a vector of the lengths of the edges of the tile • area: a positive number equal to the area of the tile • ptNum an integer equal to the number of the point determining the tile. Note that this is the number of the point in the *original* sequence of points that were tessellated.
allEdgeCounts	An integer vector of the number of edges for each of the tiles.
tabEdgeCounts	A table of allEdgeCounts.
allEdgeLengths	A vector of all of the tile edge lengths; a catenation of the edgeLengths components of the entries of indivTiles. Note that there will be many duplicate lengths since each tile edge is, in general, an edge of *two* tiles.
Areas	A vector of the areas of the tiles.
uniqueEdgeLengths	A vector of the lengths of the tiles edges with the duplicates (which occur in allEdgeLengths) being eliminated. Each tile edge is represented only once.
perimeters	A list, as returned by tilePerim() containing the perimeters of the tiles, as well as the total and the mean of these perimeters. In addition perimeters has a component perComps giving the breakdown of the perimeters into the perimeters of the parts into which tiles may have been subdivided by the clipping process.

Remark

There is a print() method for class "tileInfo" which produces a convenient display of the information returned by this function.

Author(s)

Rolf Turner <r.turner@auckland.ac.nz>

See Also

deldir() tile.list() print.tileInfo() tilePerim()

Examples

```
set.seed(42)
x    <- runif(20)
y    <- runif(20)
dxy  <- deldir(x,y,rw=c(0,1,0,1))
ixy1 <- tileInfo(dxy)
ixy2 <- tileInfo(dxy,bndry=TRUE)
if(require(polyclip)) {
    CP <- list(x=c(0.49,0.35,0.15,0.20,0.35,0.42,
                   0.43,0.62,0.46,0.63,0.82,0.79),
               y=c(0.78,0.86,0.79,0.54,0.58,0.70,
                   0.51,0.46,0.31,0.20,0.37,0.54))
    ixy3 <- tileInfo(dxy,clipp=CP) # 10 tiles are retained; the third tile,
                                   # corresponding to point 6, is
                                   # subdivided into two parts.
}
```

---

tilePerim *Calculate tile perimeters.*

Description

Calculates the perimeters of all of the Dirichlet (Voronoi) tiles in a tessellation of a set of planar points. Also calculates the sum and the mean of these perimeters.

Usage

```
tilePerim(object,inclbdry=TRUE)
```

Arguments

object	An object of class tile.list (as produced by tile.list() specifying the Dirichlet (Voronoi) tiles in a tessellation of a set of planar points.
inclbdry	Logical scalar. Should boundary segments (edges of tiles at least one of whose endpoints lies on the enclosing rectangle rw (see deldir()) be included in the perimeter?

Value

A list with components

perimeters	A vector consisting of the values of the perimeters of the Dirichlet tiles in the tessellation.
totalPerim	The sum of perimeters.

`meanPerim` The mean of `perimeters`.

`perComps` A list whose entries are vectors consisting of the "components" of the perimeters of each tile. If/when the tiles are clipped, some tiles may be subdivided by the clipping into discontiguous parts. The components referred to above are the perimeters of this parts. If no subdivision has occurred then the vector in question has a single entry equal to the perimeter of the corresponding tile. If subdivision has occurred then the perimeter of the tile is the sum of the perimeters of the components.

Note

Function added at the request of Haozhe Zhang.

Author(s)

Rolf Turner <r.turner@auckland.ac.nz>

See Also

`tile.list()`, `plot.tile.list()`

Examples

```
x <- runif(20)
y <- runif(20)
z <- deldir(x,y,rw=c(0,1,0,1))
w <- tile.list(z)
p1 <- tilePerim(w)
p0 <- tilePerim(w,inclbdry=FALSE)
p1$totalPerim - p0$totalPerim # Get 4 = the perimeter of rw.
ss <- apply(as.matrix(z$dirsgs[,1:4]),1,
            function(x){(x[1]-x[3])^2 + (x[2]-x[4])^2})
2*sum(sqrt(ss)) - p0$totalPerim # Get 0; in tilePerim() each interior
                               # edge is counted twice.
if(require(polyclip)) {
    CP <- list(x=c(0.49,0.35,0.15,0.20,0.35,0.42,
                   0.43,0.62,0.46,0.63,0.82,0.79),
               y=c(0.78,0.86,0.79,0.54,0.58,0.70,
                   0.51,0.46,0.31,0.20,0.37,0.54))
    wc <- tile.list(z,clipp=CP)
    p2 <- tilePerim(wc) # Doesn't matter here if inclbdry is TRUE or FALSE.
    p2$perComps[["pt.6"]] # The tile for point 6 has got subdivided into
                          # two parts, a tetrahedron and a triangle.
    cul <- rainbow(10)[c(1,7,3:6,2,8:10)] # Rearranging colours to improve
                                          # the contrast between contiguous tiles.
    plot(wc,number=TRUE,fillcol=cul)
}
```

toyPattern *A toy marked point pattern object, with 59 points.*

Description

A simulated object of class "ppp" provided for use in an example illustrating the application of deldir() to point pattern objects.

Usage

```
toyPattern
```

Format

An object of class "ppp" consisting of a simulated marked point pattern. Entries include

x	Cartesian x-coordinates
y	Cartesian y-coordinates
marks	factor with levels "a","b","c","d"

Source

Simulated.

Examples

```
dtoy    <- deldir(toyPattern) # "Tags" are the marks of the pattern.
set.seed(42)
dtoy.nt <- deldir(toyPattern,z=round(runif(59),2)) # Tags are numeric.
```

triang.list *Create a list of Delaunay triangles*

Description

From an object of class "deldir" produces a list of the Delaunay triangles in the triangulation of a set of points in the plane.

Usage

```
triang.list(object)
```

Arguments

object	An object of class "deldir" as produced by deldir().

Value

A list each of whose components is a 3×3 or 3×4 data frame corresponding to one of the Delaunay triangles specified by "object". The rows of each such data frame correspond to the vertices of the corresponding Delaunay triangle. The columns are:

- `ptNum` (the index of the point in the original sequence of points that is being triangulated. Note that if a point is one of a set of *duplicated* points then `ptNum` is the *first* of the indices of the points in this set.)
- `x` (the x-coordinate of the vertex)
- `y` (the y-coordinate of the vertex)
- `z` (the "auxiliary value" or "tag" `z` associated with the vertex; present only if such values were supplied in the call to `deldir()`)

The returned value has an attribute "rw" consisting of the enclosing rectangle of the triangulation.

Warning

There may not actually **be** any triangles determined by `object`, in which case this function returns an empty list with an `"rw"` attribute. See **Examples**.

Note

The code of this function was taken more-or-less directly from code written by Adrian Baddeley for the "delaunay()" function in the "spatstat" package.

Author(s)

Rolf Turner <r.turner@auckland.ac.nz>

See Also

`deldir()`, `plot.triang.list()`, `tile.list()`, `plot.tile.list()`

Examples

```
set.seed(42)
x  <- runif(20)
y  <- runif(20)
z  <- sample(1:100,20)
d1 <- deldir(x,y,z=z)
t1 <- triang.list(d1)
# A "triangulation" with no triangles!
d2 <- deldir(x=1:10,y=11:20)
plot(d2)
t2 <- triang.list(d2)
plot(t2,showrect=TRUE,rectcol="blue") # Pretty boring!
```

`triMat` *Produce matrix of triangle vertex indices.*

Description

Lists the indices of the vertices of each Delaunay triangle in the triangulation of a planar point set. The indices are listed (in increasing numeric order) as the rows of an $n \times 3$ matrix where n is the number of Delaunay triangles in the triangulation.

Usage

```
triMat(object)
```

Arguments

`object`	An object of class `deldir` (as produced by the function `deldir()`) specifying the Delaunay triangulation and Dirichlet (Voronoi) tessellation of a planar point set.

Details

This function was suggested by Robin Hankin of the School of Mathematical and Computing Sciences at Auckland University of Technology.

Value

An $n \times 3$ matrix where n is the number of Delaunay triangles in the triangulation specified by `object`. The i^{th} row consists of the indices (in the original list of points being triangulated) of vertices of the i^{th} Delaunay triangle. The indices are listed in increasing numeric order in each row.

Note

Earlier versions of this function (prior to release 0.1-14 of **deldir**) could sometimes give incorrect results. This happened if the union of three contiguous Delaunay triangles happened to constitute another triangle. This latter triangle would appear in the list of triangles produced by `triMat()` but is *not* itself a Delaunay triangle. The updated version of `triMat()` now checks for this possibility and gives (*I think!*) correct results.

Many thanks to Jay Call, who pointed out this bug to me.

Author(s)

Rolf Turner <r.turner@auckland.ac.nz>

See Also

`deldir()` `triang.list()` `plot.triang.list()`

Examples

```
# These are the data used by Jay Call to illustrate the bug
# that appeared in a previous incarnation of triMat.
xy <- data.frame(
          x = c(0.048,0.412,0.174,0.472,0.607,0.565,0.005,0.237,0.810,0.023),
          y = c(0.512,0.928,0.955,0.739,0.946,0.134,0.468,0.965,0.631,0.782)
      )
dxy <- deldir(xy)
M   <- triMat(dxy)
plot(dxy,wlines="triang",num=TRUE,axes=FALSE,cmpnt_col=c(1,1,1,1,2,1))
# The triangle with vertices {4,5,8} was listed in the output of
# the previous (buggy) version of triMat().  It is NOT a Delaunay
# triangle and hence should NOT be listed.
```

`volTriPoints` *Solute plume concentration data set.*

Description

Example solute plume concentration data set associated with the GWSDAT ("GroundWater Spatiotemporal Data Analysis Tool") project https://protect-au.mimecast.com/s/demRC91WzLH6qo3TorzN7?domain=gwsdat.net. The `deldir` package is used in this project as part of a numerical routine to estimate plume quantities (mass, average concentration and centre of mass).

Usage

```
volTriPoints
```

Format

A data frame with 232 observations on the following 3 variables.

- `x` The x-coordinates of the centres of mass of the plumes.
- `y` The y-coordinates of the centres of mass of the plumes.
- `z` The plume concentrations.

Details

This data set played a critical role in revealing a bug in the Fortran code underlying the `deldir()` function.

Source

These data were kindly provided by Wayne W. Jones of the GWSDAT project. The data set was originally named `Vol.Tri.Points`; the name was changed so as to be more consistent with my usual naming conventions.

References

Jones, W. R., Spence, M. J., Bowman, A. W., Evers, L. and Molinari, D. A. (2014). A software tool for the spatiotemporal analysis and reporting of groundwater monitoring data. *Environmental Modelling & Software* **55**, pp. 242–249.

Examples

```
dvtp <- deldir(volTriPoints)
plot(dvtp)
```

`which.tile` *Determine the tile containing a given point.*

Description

Finds the Dirichlet/Voronoi tile, of a tessellation produced by `deldir()`, that contains a given point.

Usage

```
which.tile(x, y, tl)
```

Arguments

`x`	The `x` coordinate of the point in question.
`y`	The `y` coordinate of the point in question.
`tl`	A tile list, as produced by the function `tile.list()` from a tessellation produced by `deldir()`.

Details

Just minimises the distance from the point in question to the points of the pattern determining the tiles.

Value

An integer equal to the index of the tile in which the given point lies.

Author(s)

Rolf Turner <r.turner@auckland.ac.nz>

See Also

`tile.list()` `deldir()`.

Examples

```
set.seed(42)
x   <- runif(20,0,100)
y   <- runif(20,0,100)
dxy <- deldir(x,y)
txy <- tile.list(dxy)
i   <- which.tile(30,50,txy) # The value of i here is 14.
plot(txy,showpoints=FALSE)
text(x,y,labels=1:length(txy),col="red")
points(30,50,pch=20,col="blue")
```

Index