Package 'contoureR'

August 25, 2015

Type Package

Title Contouring of Non-Regular Three-Dimensional Data

Version 1.0.5

Date 2015-08-10

Author Nicholas Hamilton

Maintainer Nicholas Hamilton <n.hamilton@unsw.edu.au>

Description
Create contour lines for a non regular series of points, potentially from a non-regular canvas.

License GPL (>= 2)

SystemRequirements C++11

Depends geometry

Imports Rcpp (>= 0.11.5), reshape, plyr

Suggests ggplot2

LinkingTo Rcpp

URL http://contoureR.com

Collate 'contoureR-package.R' 'RcppExports.R' 'RcppExports-Doc.R'
'convexHull.R' 'delaunayMesh.R' 'contourLines.R'
'orderPoints.R'

NeedsCompilation yes

Repository CRAN

Date/Publication 2015-08-25 09:09:41

R topics documented:

---

contoureR *contoureR: Contouring of Non-Regular Three-Dimensional Data*

Description

Create contour lines for a non regular series of points, potentially from a non-regular canvas.

Details

The contoureR package executes linear interpolation on a delaunay triangulated mesh strung between three-dimensional (3D) points supplied by the user. Contours are calculated across the surface constrained by the convex hull of the supplied data.

Usually, the well known functions such as contourLines from the grDevices package, expect (or rather, require) data to be regular, this means that a rectangular array or matrix of x and y coordinate pairs, each with a corresponding z value is to be modelled – that is to say the cartesian product of a numeric vector of x values of length n, with a numeric vector of y values having length m, used to produce a set of (m x n) unique points that have been concurrently provided with exactly (m x n) z values.

By restricting values to the above format, this in turn limits the region of analysis to square/rectangular canvasses (ie plane defined by geometric and orthogonal vectors parallel to the x and y axes and range bound by the [xmin,xmax] and [ymin,ymax] in the above x and y input numeric vectors, respectively). This restriction, from time-to-time, can be very inconvenient, and is a primary objective and purpose for the creation of this package.

As suggested in the previous paragraph, the contoureR package, on the other hand, has no such orthogonality / regularity requirement and can therefore be applied over obscurely shaped regions such as triangles, circles, polygons and the like. To demonstrate this, in the example provided on the current page, an equation is contoured, where firstly the x and y data is randomly selected (non regular), and then the set of values is subsequently constrained by a bounding (limiting) circle.

Note, for the moment, the only restriction is that for polygon-type regions to be modelled, then these regions must **not** have holes, since these will be filled coarsely when the Deleaunaymesh gets generated, however, in future revisions, this obstacle should be easily addressed via parameter defining a manual exclusion list of points.

See Also

getContourLines, contourLinesR and contourWalker

Examples

```
# Contour Lines for a Function, Constrained to a limited domain
# Example of the provision of non-regular data
library(contoureR)
library(ggplot2)
a  = -2; b = +2; n  = 150
x  = runif(n*n,a,b)
```

```
y  = runif(n*n,a,b)
df = data.frame(x,y)
df$z   = with(df,-x*y*exp(-x^2-y^2))
df.sub = subset(df,x^2 + y^2 < 2)
df.cnt = getContourLines(df.sub,nlevels=20)
ggplot(data=df.cnt,aes(x,y,group=Group,colour=z)) + geom_path() + theme_bw()
```

`contourLinesR` *Get Contour Lines (list)*

Description

A wrapper to the `getContourLines` function, provided to ease the transition from the `contourLines` function as part of `grDevices`

Usage

```
contourLinesR(x, y, z, nlevels = 10, levels = pretty(range(z, na.rm =
  TRUE)), ...)
```

Arguments

`x`	Numeric data for x and y coordinate, a single matrix or data-frame object can be provided for `x`, which will be used in preference to the y and z arguments. These do **NOT** need to be in any particular order nor do they need to be regular.
`y`	Numeric data for x and y coordinate, a single matrix or data-frame object can be provided for `x`, which will be used in preference to the y and z arguments. These do **NOT** need to be in any particular order nor do they need to be regular.
`z`	numeric Data for z coordinate (the coordinate to model)
`nlevels`	An integer number of bins to split the data into **iff** `levels` or `binwidth` have not been specified.
`levels`	A numeric vector of the explicitly specified levels (z values) to contour, by specifying this argument, it will override `nlevels` and/or `binwidth`. If this argument is provided, the stacking order of the contours will be preserved in the order of first occurence within the supplied vector.
`...`	any other parameters passed through to `getContourLines`

Details

This function returns data in the same format/structure as `contourLines`, ie, list of lists, which is different from the (preferred) `dataframe` object returned by the `getContourLines` function as part of the present work.

Value

A list of contours is returned, Each contour is a list with the elements:

`level`	The contour of the level
`x`	The x-coordinates of the contour
`y`	The y-coordinates of the contour

See Also

`getContourLines`

Examples

```
library(contoureR)
library(ggplot2)
x = runif(100)
y = runif(100)
df = expand.grid(x=x,y=y)
z  = with(df,x+y)
result = contourLinesR(df$x,df$y,z)
```

`contourWalker` *Contour Walker Function, Rcpp Interface to C++ Routine*

Description

This function is the R interface to the C++ core contour-walker function, totally essential for this package.

Arguments

`dm`	the n x 3 matrix representing the indexes of the vertices of the triangles in the delaunay mesh. No values should be greater than the number of rows in `xyz`.
`xyz`	the m x 3 matrix of xyz coordinates for all the points in the data.
`levels`	a numeric vector of levels to contour.
`maximumPertubation`	the maximum pertubation amount (positive or negative) as a percentage.

Details

The underlying C++ routine establishes a pointer-driven adjacency-network within each of the triangles (Dels) in the supplied Delaunay mesh, that is to say that Two Del's are deemed to be 'networked' with each other if they share adjacent edges (Edges), which are drawn between two points (Nodes). Dels are deemed as 'fully-networked' if they hold reciprocating pointers with exactly three (3) other Dels. Dels can be partially networked if one or two of the available pointers remain unassigned, which may be the case if the particular Del is a participating member of the convex hull.

Because C++ uses zero-based indexing, whilst R uses 1-based indexing, should the Delaunay Mesh (dm) be provided where the minimum value is 1, then it will be deemed to be a 1-based set and reduced accordingly.

Prior to traversing the mesh, in order to prevent degeneracies, should any Dels in the mesh have vertices which are of the same z value, and/or equal to one of the intended contouring levels, then these nodes will be pertubated (along the z direction) by an infitesimally small amount. The consequences of this approach is that when interpolating along the edges of the Del, the path will always leave at some point (if even trivially small) inbetween two (2) nodes – the path will NEVER leave directly through a node, which would otherwise lead to potential confusion as to the appropriate recipient of the path under such circumstance.

This function is not particularly convenient, and a more convenient wrapper has been produced, with all the usual checks and balances, for further information, see the getContourLines function.

Value

matrix with 6 columns: LevelID, GroupID, PathID, x, y and z

See Also

getContourLines

Examples

```
n = 100
x = runif(n)
y = runif(n)
df = expand.grid(x,y);
colnames(df) = c("x","y")
df$z = df$x^2 + df$y^2
dm  = getDelaunayMesh(df$x,df$y)
res = contourWalker(dm,as.matrix(df),levels=pretty(df$z,n=20))
res = data.frame(res); colnames(res) = c('LID','GID','PID','x','y','z')
res$Group = interaction(res$LID,res$GID)
library(ggplot2)
ggplot(res,aes(x,y,group=Group,colour=z)) + geom_path()
```

---

convexHullAM *Convex Hull via Andrews Monotone, Rcpp Interface to C++ Routine*

Description

This function is the R interface to the C++ implementation of Andrews Monotone, a well known algorithm for solving the convex hull in O(nlogn) time complexity.

Usage

```
convexHullAM_Indexes(x, y, includeColinear=FALSE,zeroBased = TRUE)
convexHullAM_Points(x, y,includeColinear=FALSE)
```

Arguments

x	NumericVector of x values
y	NumericVector of y values
includeColinear	whether to inlude points that line **ON** the hull, by default this is set to FALSE, as this is the true definition of the convex hull.
zeroBased	Whether the return indexes should be zero based (true, for use in C++), or One-Based (false, for use in R).

Value

`convexHullAM_Indexes` returns an integer vector of the indexes of the points, whilst `convexHullAM_Points` returns an `n x 2` matrix of the points themselves.

Examples

```
library(contoureR)
library(ggplot2)
set.seed(1)
x  = runif(100)
y  = runif(100)
ch = convexHullAM_Indexes(x,y,includeColinear=FALSE,zeroBased = FALSE)
ggplot(data.frame(x,y),aes(x,y)) +
 geom_point() +
 geom_path(data=data.frame(x,y)[ch,],colour="red")
```

---

getContourLines *Get Contour Lines (data.frame)*

Description

The following routine produces contour lines for a set of non-regular `x,y` and `z` values. via utilizing a Deleaunay Mesh strung between the supplied `x,y` coordinates in order to produce iso-contour data representing the third variable, `z`. To this end, by using a Deleaunay mesh, this routine does not require regular `x` and `y` data, although it can be expected to yield 'better' result, with regular / fine-grained data.

Usage

```
getContourLines(x, y, z, nlevels = 10, binwidth, levels, criticalRatio = 5)
```

Arguments

x,y	Numeric data for x and y coordinate, a single matrix or data-frame object can be provided for x, which will be used in preference to the y and z arguments. These do **NOT** need to be in any particular order nor do they need to be regular.
z	numeric Data for z coordinate (the coordinate to model)

nlevels	An integer number of bins to split the data into **iff** levels or binwidth have not been specified.
binwidth	The desired width of the bins, if specified, will override nlevels.
levels	A numeric vector of the explicitly specified levels (z values) to contour, by specifying this argument, it will override nlevels and/or binwidth. If this argument is provided, the stacking order of the contours will be preserved in the order of first occurence within the supplied vector.
criticalRatio	When producing the Delaunay Mesh, the quality of the mesh can be poor in the proximity to the convex hull, Del's that have an aspect ratio greater than this value are not considered when producing the contours. In this context, the aspect ratio is defined as the circumradius to twice its inradius, equilateral triangles have an aspect ratio of 1, everything else is larger

Value

For the function getContourLines(...), the return object is a data.frame obect representing the contours, assembled in such way to permit easy use within the ggplot2 paradigm. Such data frame contains seven (7) columns:

LID	A number representing the level
GID	Within each level, a number representing the contour group
PID	Within each group, a number representing the path/segment
x	The x-coordinates of the contour
y	The y-coordinates of the contour
z	The z-coordinates of the contour (ie value of the level)
Group	The unique identifyer for each independent contour path, calculated as being the interaction between LID and GID

See Also

contourLinesR, getDelaunayMesh and getConvexHull This is a wrapper to the C++ interface function, contourWalker.

Examples

```
# Contour Lines for Volcano Data
library(ggplot2)
data(volcano)
x = 1:nrow(volcano)
y = 1:ncol(volcano)
z = expand.grid(x=x,y=y); z$z = apply(z,1,function(xx){ volcano[ xx[1],xx[2] ]} )
df = getContourLines(z)
ggplot(df,aes(x,y,group=Group,colour=z)) + geom_path()

# Contour Lines for a Function
library(ggplot2)
a      = -2; b = 2; n = 75
x  = y = seq(a,b,by=diff(c(a,b))/(n+1))
```

```
df     = expand.grid(x=x,y=y)
df$z   = with(df,-x*y*exp(-x^2-y^2))
df.cnt = getContourLines(df)
ggplot(data=df.cnt,aes(x,y,group=Group,colour=z)) + geom_path()
```

`getConvexHull` *Get Convex Hull of Points*

Description

Returns the sequence of indexes within the supplied numeric vectors `x` and `y`, that describe the convex hull containing those points. This is a (slightly modified) implementation of the Andrews Monotone Chain, which is a well known algorithm that is able to solve the convex hull with `O(nlogn)` complexity. Typical computation time on a Macbook Air, 1.7Ghz I7, 8Gb Ram, using random points in the range [0,1]:

- 100K points 0.03 Seconds
- 1M points 0.3 seconds, and
- 10M points3.3 seconds.

Usage

```
getConvexHull(x, y, includeColinear = FALSE)
```

Arguments

`x`	numeric vector of x values
`y`	numeric vector of y values of same length as x
`includeColinear`	keep or discard the points that lie **ON** the hull, default is to discard (ie dont keep colinear points), as this is the true definition of the convex hull.

Value

Returns a vector of integers that represent the '1-based' indexes of the points relative to the `x` and `y` input arguments. The resulting vector represents the **closed** list, meaning that the first index and the last index in the series will be the same.

References

https://en.wikibooks.org/wiki/Algorithm_Implementation/Geometry/Convex_hull/Monotone_chain

Examples

```
#Generate the Convex Hull of a Series of Points
set.seed(1)
x  = runif(100)
y  = runif(100)
ch = getConvexHull(x,y)

#To demonstrate, Lets view the hull
library(ggplot2)
df = data.frame(x,y)
ggplot(data=df,aes(x,y)) +
   geom_path(data=df[ch,]) +
   geom_point()
```

`getDelaunayMesh` *Get Delaunay Mesh*

Description

Retrieves the Delaunay Mesh for a series of x and y points in 2D. With the exception of a few brief checks, is almost a direct wrapper to the `delaunayn` function as part of the geometry package.

Usage

```
getDelaunayMesh(x, y)
```

Arguments

x	numeric vector of x values
y	numeric vector of y values of same length as x

Value

`matrix` object having three columns that represent the (1-based) indexes of each vertex relative to the data in the `x` and `y` input parameters.

Examples

```
#Generate a sample Delaunay Mesh
set.seed(1)
x  = runif(100)
y  = runif(100)
dm = getDelaunayMesh(x,y)

#To demonstrate, Lets view the mesh
library(ggplot2)
library(reshape)
df = as.data.frame(dm); df$id = 1:nrow(df); df = melt(df,id="id")
df = cbind(df,data.frame(x,y)[df$value,])
```

```
ggplot(data=df,aes(x,y,group=id)) +
 geom_polygon(aes(fill=id),color="gray")
```

---

orderPoints *Order Points Clockwise or Counter-Clockwise*

Description

Returns the indexes of supplied points, x and y, ordered either clockwise or anticlockwise about another point, which by default is taken to be the non-weighted midpoint of the supplied data

Usage

```
orderPoints(x, y, ..., xm = mean(range(x)), ym = mean(range(y)),
  clockwise = TRUE)
```

Arguments

x	numeric vector of x values
y	numeric vector of y values of same length as x
...	not used
xm	the x value of the reference point
ym	the y value of the reference point
clockwise	order in clockwise or anticlockwise manner

Examples

```
#Generate a random set of points and put them clockwise order
set.seed(1)
x  = runif(100)
y  = runif(100)
op    = orderPoints(x,y)

#To demonstrate, Lets view the points in order
library(ggplot2)
df    = data.frame(x,y)
df    = df[op,];
df$id = 1:nrow(df)
ggplot(data=df,aes(x,y,colour=id)) +
    geom_path() + geom_point() +
    scale_colour_gradient(low="green",high="red")
```

Index