Package 'PBSmapping'

July 21, 2025

Version 2.74.1

Date 2024-09-25

Title Mapping Fisheries Data and Spatial Analysis Tools

Author Jon T. Schnute [aut],
Nicholas Boers [aut],
Rowan Haigh [aut, cre],
Alex Couture-Beil [ctb],
Denis Chabot [ctb],
Chris Grandin [ctb],
Alan Murta [ctb],
Angus Johnson [ctb],
Paul Wessel [ctb],
Franklin Antonio [ctb],
Nicholas J. Lewin-Koh [ctb],
Roger Bivand [ctb],
Sean Anderson [ctb]

Maintainer Rowan Haigh <rowan.haigh@dfo-mpo.gc.ca>

Copyright 2003-2024, Fisheries and Oceans Canada

Depends R (>= 3.5.0)

Suggests foreign, deldir

NeedsCompilation yes

Description This software has evolved from fisheries research conducted at the Pacific Biological Station (PBS) in 'Nanaimo', British Columbia, Canada. It extends the R language to include two-dimensional plotting features similar to those commonly available in a Geographic Information System (GIS). Embedded C code speeds algorithms from computational geometry, such as finding polygons that contain specified point events or converting between longitude-latitude and Universal Transverse Mercator (UTM) coordinates. Additionally, we include 'C++' code developed by Angus Johnson for the 'Clipper' library, data for a global shoreline, and other data sets in the public domain. Under the user's R library directory '.libPaths()', specifically in './PBSmapping/doc', a complete user's guide is offered and should be consulted to use package functions effectively.

License GPL (>= 2)

URL <https://github.com/pbs-software/pbs-mapping>,

   <https://github.com/pbs-software/pbs-mapx>,

   <https://www.angusj.com/clipper2/Docs/Overview.htm>

Repository CRAN

Date/Publication 2024-09-25 22:30:02 UTC

Contents

---

addBubbles   *Add Bubbles to Maps*

Description

Add bubbles proportional to some `EventData`'s Z column (e.g., catch or effort) to an existing plot, where each unique `EID` describes a bubble.

Usage

```
addBubbles(events, type=c("perceptual","surface","volume"),
   z.max=NULL, min.size=0, max.size=0.8, symbol.zero="+",
   symbol.fg=rgb(0,0,0,0.60), symbol.bg=rgb(0,0,0,0.30),
   legend.pos="bottomleft", legend.breaks=NULL,
   show.actual=FALSE, legend.type=c("nested","horiz","vert"),
   legend.title="Abundance", legend.cex=.8, neg.col="RYB", ...)
```

Arguments

events	[EventData](#) to use (*required*).
type	`character` – scaling option for bubbles where `"perceptual"` emphasizes large z-values, `"volume"` emphasizes small z-values, and `"surface"` lies in between.
z.max	`numeric` – maximum value for z (default = `max(events$Z)`); determines the largest bubble; keeps the same legend for different maps.
min.size	`numeric` – minimum size (inches) for a bubble representing `min(events$Z)`. The legend may not actually include a bubble of this size because the calculated legend.breaks does not include the `min(events$Z)`.
max.size	`numeric` – maximum size (inches) for a bubble representing `z.max`. A legend bubble may exceed this size when `show.actual` is `FALSE` (on account of using `pretty(...)`).
symbol.zero	`character` – symbol to represent z-values equal to 0.
symbol.fg	`character` – bubble outline (border) colour.
symbol.bg	`character` – bubble interior (fill) colour. If a vector, the first element represents `min(legend.breaks)` and the last element represents `max(legend.breaks)`; colours are interpolated for values of `events$Z` between those boundaries. For values outside of those boundaries, interiors remain unfilled.
legend.pos	`character|numeric` – position for the legend.
legend.breaks	`numeric` – break values for categorizing the z-values. The automatic method should work if zeroes are present; otherwise, you can specify your own break values for the legend. If a single number, specifies the number of breaks; if a vector, specifies the breaks.
show.actual	`logical` – if `FALSE`, legend values are obtained using `pretty(...)`, and consequently, the largest bubble may be larger than `z.max`; if `TRUE`, the largest bubble in the legend will correspond to `z.max`.

legend.type	character – display format for legend.
legend.title	character – title for legend.
legend.cex	numeric – size of legend text.
neg.col	character – bubble colour for any potential negative values.
...	dots – additional arguments for points function that plots zero-value symbols.

Details

Modified from (and for the legend, strongly inspired by) Tanimura et al. (2006) by Denis Chabot to work with **PBSmapping**.

Furthermore, Chabot's modifications make it possible to draw several maps with bubbles that all have the same scale (instead of each bubble plot having a scale that depends on the maximum z-value for that plot). This is done by making z.max equal to the largest z-value from all maps that will be plotted.

The user can also add a legend in one of four corners (see legend) or at a specific c(X,Y) position. If legend.pos is NULL, no legend is drawn.

Author(s)

Denis Chabot, Research Scientist
Maurice-Lamontagne Institute, Fisheries & Oceans Canada (DFO), Mont-Joli QC

Maintainer: Rowan Haigh, Program Head – Offshore Rockfish
Pacific Biological Station (PBS), Fisheries & Oceans Canada (DFO), Nanaimo BC
locus opus: Offsite, Vancouver BC
Last modified Rd: 2024-09-25

References

Tanimura, S., Kuroiwa, C., and Mizota, T. (2006) Proportional symbol mapping in R. *Journal of Statistical Software* **15**(5).

See Also

In package **PBSmapping**:
addPolys, EventData, RGB2RYB, surveyData

Examples

```
local(envir=.PBSmapEnv,expr={
  oldpar = par(no.readonly=TRUE)
  #--- common code for both examples below
  data(nepacLL,surveyData,envir=.PBSmapEnv)
  surveyData$Z <- surveyData$catch

  #--- plot a version that only varies the size
  plotMap(nepacLL, xlim=c(-131.8,-127.2), ylim=c(50.5,52.7),
    col="gainsboro",plt=c(.08,.99,.08,.99), cex.axis=1.2, cex.lab=1.5)
```

```
addBubbles(surveyData, symbol.bg=rgb(.9,.5,0,.6),
  legend.type="nested", symbol.zero="+", col="grey")

#--- plot a version that uses different symbol colours
plotMap(nepacLL, xlim=c(-131.8,-127.2), ylim=c(50.5,52.7),
col="gainsboro",plt=c(.08,.99,.08,.99), cex.axis=1.2, cex.lab=1.5)
subset <- surveyData[surveyData$Z <= 1000, ]
addBubbles(subset, symbol.bg=c("red", "yellow", "green"),
  legend.type="horiz", legend.breaks=pretty(range(subset$Z), n=11),
  symbol.zero=FALSE, col="grey", min.size=0.1, max.size=0.4)
par(oldpar)
})
```

---

addCompass   *Add Compass Rose to Map*

Description

Add a compass rose to an existing map, similar to those found on nautical charts showing both true north and magnetic north.

Usage

```
addCompass(X, Y, rot="magN", useWest=TRUE, year,
  cex=1, col.compass=c("gainsboro","blue","yellow","black"), ...)
```

Arguments

X	numeric – longitude coordinate (degrees N) for centroid of compass rose.
Y	numeric – latitude coordinate (degrees W) for centroid of compass rose.
rot	numeric|character – Rotation (degrees) counterclockwise from 0 degrees (true North, see details).
useWest	logical – if TRUE, assumes longitude coordinates are centered on Greenwich extending westward to -180 and eastward to 180.
year	numeric – specific year to use from IGRF-13 (see details).
cex	numeric – character expansion to use in the display.
col.compass	character – colours for compass rose components (in order): 1=background compass, 2=rotated arms, 3=central button, 4=pch (border).
...	dots – additional parameters to pass to the text function.

Details

The basic idea comes from Jim Lemon (see References), but is modified here to reflect a compass rose used on BC nautical charts.

The default rotation ("magN") is a calculation of the initial bearing of a great-circle arc from the compass position to the north geomagnetic rot using the function `calcGCdist`. The default year

is the current year, but the user can choose years from 1900 to 2025 for approximate rot locations using model output from NOAA's IGRF-13 (International Geomagnetic Reference Field).

The user can also specify a fixed rotation (e.g. `rot=-30`) or no rotation (either `rot=0` or `rot="trueN"`).

Value

No value returned.

Author(s)

Rowan Haigh, Program Head – Offshore Rockfish
Pacific Biological Station (PBS), Fisheries & Oceans Canada (DFO), Nanaimo BC
locus opus: Regional Headquarters, Vancouver BC
Last modified Rd: `2022-07-05`

References

[R-sig-Geo] How to display a compass rose on a map

Magnetic North, Geomagnetic and Magnetic Poles

See Also

`addBubbles`, `addLabels`, `addPoints`, `addStipples`, `calcGCdist`

Examples

```
local(envir=.PBSmapEnv,expr={
  data(nepacLL,envir=.PBSmapEnv)
  par(mfrow=c(1,1),mar=c(3,4,0.5,0.5))
  plotMap(nepacLL, xlim=c(-134.5,-124.5), ylim=c(48,55), plt=NULL,
    col="lightyellow", cex.axis=1.2, cex.lab=1.5)
  addCompass(-132, 49.5, rot=-12, cex=1.5)
})
```

---

addLabels	*Add Labels to an Existing Plot*

Description

Add the `label` column of `data` to the existing plot.

Usage

```
addLabels (data, xlim = NULL, ylim = NULL, polyProps = NULL,
           placement = "DATA", polys = NULL, rollup = 3,
           cex = NULL, col = NULL, font = NULL, ...)
```

Arguments

data	EventData or PolyData to add (*required*).
xlim	range of X-coordinates.
ylim	range of Y-coordinates.
polyProps	PolyData specifying which labels to plot and their properties. par parameters passed as direct arguments supersede these data.
placement	one of ″DATA″, ″CENTROID″, ″MEAN_RANGE″, or ″MEAN_XY″.
polys	PolySet to use for calculating label placement.
rollup	level of detail at which to process polys, and it should match that in data. 1 = PIDs only, 2 = outer contours only, and 3 = no roll-up.
cex	vector describing character expansion factors (cycled by EID or PID).
col	vector describing colours (cycled by EID or PID).
font	vector describing fonts (cycled by EID or PID).
...	additional par parameters for the text function.

Details

If data is EventData, it must minimally contain the columns EID, X, Y, and label. Since the EID column does not match a column in polys, set placement = ″DATA″. The function plots each label at its corresponding X/Y coordinate.

If data is PolyData, it must minimally contain the columns PID and label. If it also contains X and Y columns, set placement = ″DATA″ to plot labels at those coordinates. Otherwise, set placement to one of ″CENTROID″, ″MEAN_RANGE″, or ″MEAN_XY″. When placement != ″DATA″, supply a PolySet polys. Using this PolySet, the function calculates a centroid, mean range, or mean X/Y coordinate for each polygon, and then links those PolyData with data by PID/SID to determine label coordinates.

If data contains both PID and EID columns, the function assumes it is PolyData and ignores the EID column.

For additional help on the arguments cex, col, and font, please see par.

Value

EventData or PolyData with X and Y columns that can subsequently reproduce the labels on the plot. Modify this data frame to tweak label positions.

Author(s)

Nicholas M. Boers, Staff Software Engineer
Jobber, Edmonton AB
Last modified Rd: 2013-04-10

See Also

addPoints, calcCentroid, calcMidRange, calcSummary, EventData, plotPoints, PolyData.

Examples

```
local(envir=.PBSmapEnv,expr={
  oldpar = par(no.readonly=TRUE)
  #--- create sample PolyData to label Vancouver Island
  labelData <- data.frame(PID=33, label="Vancouver Island");
  #--- load data
  if (!is.null(version$language) && (version$language == "R"))
    data(nepacLL,envir=.PBSmapEnv)
  #--- plot the map
  plotMap(nepacLL,xlim=c(-129,-122.6),ylim=c(48,51.1),col="lemonchiffon")
  #--- add the labels
  addLabels(labelData,placement="CENTROID",polys=nepacLL,cex=1.2,col=2,font=2)
  par(oldpar)
})
```

---

addLines   *Add a PolySet to an Existing Plot as Polylines*

Description

Add a [PolySet](#) to an existing plot, where each unique (PID, SID) describes a polyline.

Usage

```
addLines (polys, xlim = NULL, ylim = NULL,
    polyProps = NULL, lty = NULL, col = NULL, arrows = FALSE, ...)
```

Arguments

polys	[PolySet](#) to add (*required*).
xlim	range of X-coordinates.
ylim	range of Y-coordinates.
polyProps	[PolyData](#) specifying which polylines to plot and their properties. [par](#) parameters passed as direct arguments supersede these data.
lty	vector of line types (cycled by PID).
col	vector of colours (cycled by PID).
arrows	Boolean value; if TRUE, add arrows using the [arrows](#) function and consider the arguments angle, length, and code.
...	additional [par](#) parameters for the [lines](#) function.

Details

The plotting routine does not connect the last vertex of each discrete polyline to the first vertex of that polyline. It clips polys to xlim and ylim before plotting.

For additional help on the arguments lty and col, please see [par](#).

Value

[PolyData](#) consisting of the `PolyProps` used to create the plot.

Author(s)

Nicholas M. Boers, Staff Software Engineer
Jobber, Edmonton AB
Last modified `Rd`: 2013-04-10

See Also

calcLength, clipLines, closePolys, convLP, fixBound, fixPOS, locatePolys, plotLines,
thinPolys, thickenPolys.

Examples

```
local(envir=.PBSmapEnv,expr={
  oldpar = par(no.readonly=TRUE)
  #--- create a PolySet to plot
  polys <- data.frame(PID=rep(1,4),POS=1:4,X=c(0,1,1,0),Y=c(0,0,1,1))
  polys <- as.PolySet(polys, projection=1)
  #--- plot the PolySet
  plotLines(polys, xlim=c(-.5,1.5), ylim=c(-.5,1.5), projection=1)
  #--- add the PolySet to the plot (in a different style)
  addLines(polys, lwd=5, col=3)
  par(oldpar)
})
```

---

addPoints   *Add EventData/PolyData to an Existing Plot as Points*

Description

Add [EventData](#)/[PolyData](#) to an existing plot, where each unique `EID` describes a point.

Usage

```
addPoints (data, xlim = NULL, ylim = NULL, polyProps = NULL,
           cex = NULL, col = NULL, pch = NULL, ...)
```

Arguments

data	[EventData](#) or [PolyData](#) to add (*required*).
xlim	range of X-coordinates.
ylim	range of Y-coordinates.
polyProps	[PolyData](#) specifying which points to plot and their properties. [par](#) parameters passed as direct arguments supersede these data.

cex	vector describing character expansion factors (cycled by EID or PID).
col	vector describing colours (cycled by EID or PID).
pch	vector describing plotting characters (cycled by EID or PID).
...	additional par parameters for the points function.

Details

This function clips data to xlim and ylim before plotting. It only adds PolyData containing X and Y columns.

For additional help on the arguments cex, col, and pch, please see par.

Value

PolyData consisting of the PolyProps used to create the plot.

Author(s)

Nicholas M. Boers, Staff Software Engineer
Jobber, Edmonton AB
Last modified Rd: 2013-04-10

See Also

combineEvents, convDP, findPolys, locateEvents, plotPoints.

Examples

```
local(envir=.PBSmapEnv,expr={
  oldpar = par(no.readonly=TRUE)
  #--- load the data (if using R)
  if (!is.null(version$language) && (version$language=="R"))
    data(nepacLL,surveyData,envir=.PBSmapEnv)
  #--- plot a map
  plotMap(nepacLL, xlim=c(-136, -125), ylim=c(48, 57))
  #--- add events
  addPoints(surveyData, col=1:7)
  par(oldpar)
})
```

---

addPolys	*Add Polygon PolySet to Existing Plot*

Description

Add a PolySet to an existing plot, where each unique (PID, SID) describes a polygon.

Usage

```
addPolys(polys, xlim = NULL, ylim = NULL, polyProps = NULL,
        border = NULL, lty = NULL, col = NULL, colHoles = NULL,
        density = NA, angle = NULL, ...)

.insertNAs(polys, idx)

.preparePolyProps(polysPID, polysSID, polyProps)

.rollupPolys(polys, rollupMode, exteriorCCW, closedPolys, addRetrace)
```

Arguments

polys	data.frame – [PolySet](#) to add (*required*)
xlim	numeric – range of X-coordinates.
ylim	numeric – range of Y-coordinates.
polyProps	data.frame – [PolyData](#) specifying which polygons to plot and their properties. [par](#) parameters passed as direct arguments supersede these data.
border	character|numeric – vector describing edge colours (cycled by PID).
lty	character|numeric – vector describing line types (cycled by PID).
col	character|numeric – vector describing fill colours (cycled by PID).
colHoles	character|numeric – vector describing hole colours (cycled by PID). The default, NULL, should be used in most cases as it renders holes transparent. colHoles is designed solely to eliminate retrace lines when images are converted to PDF format. If colHoles is specified, underlying information (i.e., previously plotted shapes) will be obliterated. If NA is specified, only outer polygons are drawn, consequently filling holes.
density	numeric – vector describing shading line densities (lines per inch, cycled by PID).
angle	numeric – vector describing shading line angles (degrees, cycled by PID).
...	dots – additional [par](#) parameters for the [polygon](#) function.
idx	character|numeric – index of PIDs used to extract select polygons/polylines from a PolySet
polysPID	numeric – vector of primary polygon identification numbers (PIDs)
polysSID	numeric – vector of secondary polygon identification numbers (SIDs)
rollupMode	numeric – method for rolling up a PolySet; essentially controls when to introduce a new PID (or PID,SID). Options: 1 = roll-up to the PID level (only PIDs in the result) 2 = roll-up to the outer contour level (only outer contours in the result) 3 = do not roll-up
exteriorCCW	numeric – modify orientation of vertices (clockwise or counter-clockwise): -1 = do not modify 0 = exterior should be clockwise +1 = exterior should be counter-clockwise

closedPolys	numeric – whether the last and first vertices should be the same:
	-1 = do not modify
	0 = ensure polygons do not close
	+1 = close the polygons
addRetrace	numeric – determines whether to add retrace lines to the first vertex of the parent after outputting a child:
	0 = do not add
	1 = add

Details

The plotting routine connects the last vertex of each discrete polygon to the first vertex of that polygon. It supports both borders (border, lty) and fills (col, density, angle). It clips polys to xlim and ylim before plotting.

For additional help on the arguments border, lty, col, density, and angle, please see [polygon](#) and [par](#).

Value

[PolyData](#) consisting of the PolyProps used to create the plot.

Note

Auxiliary dot function '.insertNAs' facilitates (hastens) the plotting of polygons and polylines. It also reduces the incidence of retrace lines.

Auxiliary dot function '.preparePolyProps' performs at least one of the following tasks:
1) creates 'polyProps' if it equals NULL;
2) adds 'SID' column to 'polyProps' if one exists in 'polys';
3) removes from 'polyProps' any PIDs that do not exist in 'polys'.
Returns a polyProps object.

Auxiliary dot function '.rollupPolys' does not validate a PolySet; returns a rolled-up PolySet or NULL.

Author(s)

[Nicholas M. Boers](#), Software Engineer, Jobber, Edmonton AB

Maintainer: [Rowan Haigh](#), Program Head – Offshore Rockfish
Pacific Biological Station (PBS), Fisheries & Oceans Canada (DFO), Nanaimo BC
locus opus: Remote office, Vancouver BC
Last modified Rd: 2024-09-03

See Also

In package **PBSmapping**:
[addLabels](#), [addLines](#) (also uses '.preparePolyProps' and '.insertNAs'), [addPoints](#), [addStipples](#)
'.rollupPolys' is also called by:
[calcArea](#), [calcCentroid](#), [calcLength](#), [calcSummary](#), [fixPOS](#)

Examples

```
local(envir=.PBSmapEnv,expr={
  oldpar = par(no.readonly=TRUE)
  #--- create a PolySet to plot
  polys <- data.frame(PID=rep(1,4),POS=1:4,X=c(0,1,1,0),Y=c(0,0,1,1))
  polys <- as.PolySet(polys, projection=1)
  #--- plot the PolySet
  plotPolys(polys,xlim=c(-.5,1.5),ylim=c(-.5,1.5),density=0,projection=1)
  #--- add the PolySet to the plot (in a different style)
  addPolys(polys,col="green",border="blue",lwd=3)
  par(oldpar)
})
```

---

addStipples   *Add Stipples to an Existing Plot*

Description

Add stipples to an existing plot.

Usage

```
addStipples (polys, xlim=NULL, ylim=NULL, polyProps=NULL,
   side=1, density=1, distance=4, ...)
```

Arguments

polys	PolySet that provides the stipple boundaries (*required*).
xlim	range of X-coordinates.
ylim	range of Y-coordinates.
polyProps	PolyData specifying which polygons to stipple and their properties. par parameters passed as direct arguments supersede these data.
side	one of -1, 0, or 1, corresponding to outside, both sides, or inside, respectively.
density	density of points, relative to the default.
distance	distance to offset points, measured as a percentage of the absolute difference in xlim.
...	additional par parameters for the points function.

Details

This function locates stipples based on the PolySet polys and does not stipple degenerate lines.

Value

PolyData consisting of the PolyProps used to create the plot.

Author(s)

Nicholas M. Boers, Staff Software Engineer
Jobber, Edmonton AB
Last modified Rd: 2013-04-10

See Also

addPoints, addPolys, plotMap, plotPoints, plotPolys, points, PolySet.

Examples

```
local(envir=.PBSmapEnv,expr={
  oldpar = par(no.readonly=TRUE)
  #--- load the data (if using R)
  if (!is.null(version$language) && (version$language=="R"))
    data(nepacLL,envir=.PBSmapEnv)
  #--- plot a map
  plotMap(nepacLL,xlim=c(-128.66,-122.83),ylim=c(48.00,51.16))
  #--- add stippling
  addStipples(nepacLL,col="purple",pch=20,cex=0.25,distance=2)
  par(oldpar)
})
```

---

appendPolys	*Append a Two-Column Matrix to a PolySet*

Description

Append a two-column matrix to a PolySet, assigning PID and possibly SID values automatically or as specified in its arguments.

Usage

```
appendPolys (polys, mat, PID = NULL, SID = NULL, isHole = FALSE)
```

Arguments

polys	existing PolySet; if NULL, creates a new PolySet (*required*).
mat	two-column matrix to append (*required*).
PID	new polygon's PID.
SID	new polygon's SID.
isHole	Boolean value; if TRUE, mat represents a hole.

Details

If the `PID` argument is `NULL`, the appended polygon's `PID` will be one greater than the maximum within `polys` (if defined); otherwise, it will be 1.

If `polys` contains an `SID` column and the `SID` argument equals `NULL`, this function uses the next available `SID` for the corresponding `PID`.

If `polys` does not contain an `SID` column and the caller passes an `SID` argument, all existing polygons will receive an `SID` of 1. The new polygon's `SID` will match the `SID` argument.

If `isHole = TRUE`, the polygon's `POS` values will appropriately represent a hole (reverse order of `POS`).

If (`PID`, `SID`) already exists in the [PolySet](#), the function will issue a warning and duplicate those identifiers.

Value

[PolySet](#) containing `mat` appended to `polys`. The function retains attributes from `polys`.

Author(s)

[Nicholas M. Boers](#), Staff Software Engineer
Jobber, Edmonton AB
Last modified Rd: 2013-04-10

See Also

[addPolys](#), [clipPolys](#), [closePolys](#), [convLP](#), [fixBound](#), [fixPOS](#), [joinPolys](#), [plotMap](#), [plotPolys](#).

Examples

```
local(envir=.PBSmapEnv,expr={
  #--- create two simple matrices
  a <- matrix(data=c(0,0,1,0,1,1,0,1),ncol=2,byrow=TRUE);
  b <- matrix(data=c(2,2,3,2,3,3,2,3), ncol=2,byrow=TRUE);
  #--- build a PolySet from them
  polys <- appendPolys(NULL, a);
  polys <- appendPolys(polys, b);
  #--- print the result
  print (polys);
})
```

---

bcBathymetry   *Data: Bathymetry Spanning BC Coast*

Description

Bathymetry data spanning British Columbia's coast.

Usage

```
data(bcBathymetry)
```

Format

Three-element list: `x` = vector of horizontal grid line locations, `y` = vector of vertical grid line locations, `z` = (x by y) matrix containing water depths measured in meters. Positive values indicate distance below sea level and negative values above it.

The functions `'graphics::contour'` and `'grDevices::contourLines'` expect data in this format. Function convCP converts the output from `'grDevices::contourLines'` into a PolySet.

Note

In R, the data must be loaded using the `'utils::data'` function.

Author(s)

Nicholas M. Boers, Staff Software Engineer
Jobber, Edmonton AB

Maintainer: Rowan Haigh, Program Head – Offshore Rockfish
Pacific Biological Station (PBS), Fisheries & Oceans Canada (DFO), Nanaimo BC
locus opus: Regional Headquarters, Vancouver BC
Last modified Rd: `2022-07-07`

Source

Bathymetry data acquired from the Scripps Institution of Oceanography at the University of San Diego.

Using their online form, we requested bathymetry data for the complete nepacLL region. At forty megabytes, the data were not suitable for distribution in our mapping package. Therefore, we reduced the data to the range $-140° \leq x \leq -122°$ and $47° \leq y \leq 61°$.

References

Smith, W.H.F. and Sandwell, D.T. (1997) Global seafloor topography from satellite altimetry and ship depth soundings. *Science* **277**, 1957-1962.
Website: https://topex.ucsd.edu/WWW_html/mar_topo.html

See Also

In package **graphics**:
contour
In package **grDevices**:
contourLines
In package **PBSmapping**:
convCP, nepacLL, nepacLLhigh

calcArea   *Calculate the Areas of Polygons*

Description

Calculate the areas of polygons found in a PolySet.

Usage

```
calcArea (polys, rollup = 3)
```

Arguments

polys   PolySet to use.

rollup   level of detail in the results; 1 = PIDs only, by summing all the polygons with
             the same PID, 2 = outer contours only, by subtracting holes from their parent,
             and 3 = no roll-up.

Details

If rollup equals 1, the results contain an area for each unique PID only. When it equals 2, they
contain entries for outer contours only. Finally, setting it to 3 prevents roll-up, and they contain
areas for each unique (PID, SID).

Outer polygons have positive areas and inner polygons negative areas. When polygons are rolled
up, the routine sums the positive and negative areas and consequently accounts for holes.

If the PolySet's projection attribute equals "LL", the function projects the PolySet in UTM first.
If the PolySet's zone attribute exists, it uses it for the conversion. Otherwise, it computes the mean
longitude and uses that value to determine the zone. The longitude range of zone i is $-186 + 6i° <
x \leq -180 + 6i°$.

Value

PolyData with columns PID, SID (*may be missing*), and area. If the projection equals "LL" or
"UTM", the units of area are square kilometres.

Author(s)

Nicholas M. Boers, Staff Software Engineer
Jobber, Edmonton AB
Last modified Rd: 2022-09-06

See Also

calcCentroid, calcLength, calcMidRange, calcSummary, locatePolys.

Examples

```
local(envir=.PBSmapEnv,expr={
  #--- load the data (if using R)
  if (!is.null(version$language) && (version$language == "R"))
    data(nepacLL,envir=.PBSmapEnv)
  #--- convert LL to UTM so calculation makes sense
  attr(nepacLL, "zone") <- 9
  nepacUTM <- convUL(nepacLL)
  #--- calculate and print the areas
  print(calcArea(nepacUTM))
})
```

---

calcCentroid	*Calculate the Centroids of Polygons*

Description

Calculate the centroids of polygons found in a PolySet.

Usage

```
calcCentroid (polys, rollup = 3)
```

Arguments

polys	PolySet to use.
rollup	level of detail in the results; 1 = PIDs only, 2 = outer contours only, and 3 = no roll-up. When `rollup` equals 1 and 2, the function appropriately adjusts for polygons with holes.

Details

If `rollup` equals 1, the results contain a centroid for each unique PID only. When it equals 2, they contain entries for outer contours only. Finally, setting it to 3 prevents roll-up, and they contain a centroid for each unique (PID, SID).

Value

PolyData with columns PID, SID (*may be missing*), X, and Y.

Author(s)

[Nicholas M. Boers](Nicholas), Staff Software Engineer
Jobber, Edmonton AB
Last modified Rd: 2013-04-10

See Also

calcArea, calcLength, calcMidRange, calcSummary, locateEvents, locatePolys.

Examples

```
local(envir=.PBSmapEnv,expr={
  #--- load the data (if using R)
  if (!is.null(version$language) && (version$language=="R"))
    data(nepacLL,envir=.PBSmapEnv)
  #--- calculate and print the centroids for several polygons
  print(calcCentroid(nepacLL[is.element(nepacLL$PID,c(33,39,47)),]))
})
```

---

calcConvexHull	*Calculate Convex Hull for a Set of Points*

Description

Calculate the convex hull for a set of points.

Usage

```
calcConvexHull(xydata, keepExtra=FALSE)
.closestPoint(pts, pt)
```

Arguments

xydata	numeric – data frame with columns 'X' and 'Y' containing spatial coordinates
keepExtra	logical – if TRUE, retain any additional columns from the input data frame 'xydata'.
pts	numeric – data frame with columns 'X' and 'Y'
pt	numeric – data frame with columns 'X' and 'Y' and ONE row

Details

Uses the function chull() in the package grDevices. By default, it ignores all columns other than X and Y; however, the user can choose to retain additional columns in xydata by specifying keepExtra=TRUE.

Value

[PolySet](#) with columns PID, POS, X, Y, and additional columns in xydata if keepExtra=TRUE.

Auxiliary dot function '.closestPoint' returns a vector of length 'pts' where TRUE indicates that the point is closest to 'pt'. Returns several TRUE values when several points are equidistant.

Author(s)

[Nicholas M. Boers](#), Software Engineer, Jobber, Edmonton AB

Maintainer: [Rowan Haigh](#), Program Head – Offshore Rockfish
Pacific Biological Station (PBS), Fisheries & Oceans Canada (DFO), Nanaimo BC
locus opus: Remote office, Vancouver BC
Last modified Rd: 2024-09-03

See Also

In package **PBSmapping**:
addPoints, addPolys, calcArea, calcCentroid, calcMidRange, calcSummary, locateEvents, plotMap, plotPoints, plotPolys.

Examples

```
local(envir=.PBSmapEnv,expr={
  oldpar = par(no.readonly=TRUE)
  data(surveyData,envir=.PBSmapEnv)
  #--- plot the convex hull, and then plot the points
  plotMap(calcConvexHull(surveyData),col="moccasin")
  addPoints(surveyData,col="blue",pch=17,cex=.6)
  par(oldpar)
})
```

---

calcGCdist	*Calculate Great-Circle Distance*

Description

Calculate the great-circle distance between geographic (LL) coordinates. Also calculate the initial bearing of the great-circle arc (at its starting point).

Usage

```
calcGCdist(lon1, lat1, lon2, lat2, R=6371.2)
```

Arguments

lon1	numeric – Longitude coordinate (degrees) of the start point.
lat1	numeric – Latitude coordinate(degrees) of the start point.
lon2	numeric – Longitude coordinate(degrees) of the end point.
lat2	numeric – Latitude coordinate(degrees) of the end point.
R	numeric – Mean radius (km) of the Earth.

Details

The great-circle distance is calculated between two points along a spherical surface using the shortest distance and disregarding topography.

Method 1: Haversine Formula

$$a = \sin^2((\phi_2 - \phi_1)/2) + \cos(\phi_1)\cos(\phi_2)\sin^2((\lambda_2 - \lambda_1)/2)$$

$$c = 2\operatorname{atan2}(\sqrt{a}, \sqrt{1-a})$$

$$d = Rc$$

where
ϕ = latitude (in radians),
λ = longitude (in radians),
R = radius (km) of the Earth,
a = square of half the chord length between the points,
c = angular distance in radians,
d = great-circle distance (km) between two points.

Method 2: Spherical Law of Cosines

$$d = \mathrm{acos}(\sin(\phi_1)\sin(\phi_2) + \cos(\phi_1)\cos(\phi_2)\cos(\lambda_2 - \lambda_1))R$$

The initial bearing (aka forward azimuth) for the start point can be calculated using:

$$\theta = \mathrm{atan2}(\sin(\lambda_2 - \lambda_1)\cos(\phi_2), \cos(\phi_1)\sin(\phi_2) - \sin(\phi_1)\cos(\phi_2)\cos(\lambda_2 - \lambda_1))$$

Value

A list obect containing:
a – Haversine a = square of half the chord length between the points,
c – Haversine c = angular distance in radians,
d – Haversine d = great-circle distance (km) between two points,
d2 – Law of Cosines d = great-circle distance (km) between two points,
theta – Initial bearing θ (degrees) for the start point.

Note

If one uses the north geomagnetic pole as an end point, θ crudely approximates the magnetic declination.

Author(s)

Rowan Haigh, Program Head – Offshore Rockfish
Pacific Biological Station (PBS), Fisheries & Oceans Canada (DFO), Nanaimo BC
locus opus: Remote office, Vancouver BC
Last modified `Rd`: `2024-03-11`

References

Movable Type Scripts – Calculate distance, bearing and more between Latitude/Longitude points

See Also

In package **PBSmapping**:
addCompass, calcArea, calcCentroid, calcLength

Examples

```
local(envir=.PBSmapEnv,expr={
  #-- Distance between southern BC waters and north geomagnetic pole
  print(calcGCdist(-126.5,48.6,-72.7,80.4))
})
```

---

calcLength *Calculate the Length of Polylines*

Description

Calculate the length of polylines found in a [PolySet](#).

Usage

```
calcLength (polys, rollup = 3, close = FALSE)
```

Arguments

polys	[PolySet](#) to use.
rollup	level of detail in the results; 1 = PIDs only, summing the lengths of each SID within each PID, and 3 = no roll-up. Note: rollup 2 has no meaning in this function and, if specified, will be reset to 3.
close	Boolean value; if TRUE, include the distance between each polygon's last and first vertex, if necessary.

Details

If rollup equals 1, the results contain an entry for each unique PID only. Setting it to 3 prevents roll-up, and they contain an entry for each unique (PID, SID).

If the projection attribute equals "LL", this routine uses Great Circle distances to compute the surface length of each polyline. In doing so, the algorithm simplifies Earth to a sphere.

If the projection attribute equals "UTM" or 1, this routine uses Pythagoras' Theorem to calculate lengths.

Value

[PolyData](#) with columns PID, SID (*may be missing*), and length. If projection equals "UTM" or "LL", lengths are in kilometres. Otherwise, lengths are in the same unit as the input [PolySet](#).

Author(s)

[Nicholas M. Boers](#), Staff Software Engineer
Jobber, Edmonton AB
Last modified Rd: 2013-04-10

See Also

[calcArea](), [calcCentroid](), [calcMidRange](), [calcSummary](), [locatePolys]().

Examples

```
local(envir=.PBSmapEnv,expr={
  #--- load the data (if using R)
  if (!is.null(version$language) && (version$language=="R"))
    data(nepacLL,envir=.PBSmapEnv)
  #--- calculate the perimeter of Vancouver Island
  print(calcLength(nepacLL[nepacLL$PID==33, ]))
})
```

---

calcMidRange   *Calculate the Midpoint of the X/Y Ranges of Polygons*

Description

Calculate the midpoint of the X/Y ranges of polygons found in a [PolySet]().

Usage

```
calcMidRange (polys, rollup = 3)
```

Arguments

polys   [PolySet]() to use.
rollup   level of detail in the results; 1 = PIDs only, 2 = outer contours only, and 3 = no
            roll-up.

Details

If rollup equals 1, the results contain a mean range for each unique PID only. When it equals 2,
they contain entries for outer contours only. Finally, setting it to 3 prevents roll-up, and they contain
a mean range for each unique (PID, SID).

Value

[PolyData]() with columns PID, SID (*may be missing*), X, and Y.

Author(s)

[Nicholas M. Boers](), Staff Software Engineer
Jobber, Edmonton AB
Last modified Rd: 2013-04-10

See Also

[calcArea](), [calcCentroid](), [calcLength](), [calcSummary]().

Examples

```
local(envir=.PBSmapEnv,expr={
  #--- load the data (if using R)
  if (!is.null(version$language) && (version$language=="R"))
    data(nepacLL,envir=.PBSmapEnv)
  #--- calculate and print the centroids for several polygons
  print(calcMidRange(nepacLL[is.element(nepacLL$PID,c(33,39,47)),]))
})
```

---

calcSummary	*Apply Functions to Polygons in a PolySet*

Description

Apply functions to polygons in a [PolySet](#).

Usage

```
calcSummary (polys, rollup = 3, FUN, ...)
```

Arguments

polys	[PolySet](#) to use.
rollup	level of detail in the results; 1 = PIDs only, by removing the SID column, and then passing each PID into FUN, 2 = outer contours only, by making hole SIDs equal to their parent's SID, and then passing each (PID, SID) into FUN, and 3 = no roll-up.
FUN	the function to apply; it must accept a vector and return a vector or scalar.
...	optional arguments for FUN.

Details

If rollup equals 1, the results contain an entry for each unique PID only. When it equals 2, they contain entries for outer contours only. Finally, setting it to 3 prevents roll-up, and they contain an entry for each unique (PID, SID).

Value

[PolyData](#) with columns PID, SID (*may be missing*), X, and Y. If FUN returns a vector of length greater than 1 (say *n*), names the columns X1, X2, ..., X*n* and Y1, Y2, ..., Y*n*.

Author(s)

[Nicholas M. Boers](#), Staff Software Engineer
Jobber, Edmonton AB
Last modified Rd: 2013-04-10

See Also

[calcArea](), [calcCentroid](), [calcConvexHull](), [calcLength](), [calcMidRange](), [combineEvents](), [findPolys](),
[locateEvents](), [locatePolys](), [makeGrid](), [makeProps]().

Examples

```
local(envir=.PBSmapEnv,expr={
  #--- load the data (if using R)
  if (!is.null(version$language) && (version$language=="R"))
    data(nepacLL,envir=.PBSmapEnv)
  #--- calculate and print the centroids for several polygons
  print(calcSummary(nepacLL[is.element(nepacLL$PID,c(33,39,47)),],
    rollup=3, FUN=mean))
})
```

---

calcVoronoi   *Calculate Voronoi Tesselation*

Description

Calculate the Voronoi (Dirichlet) tesselation for a set of points.

Usage

```
calcVoronoi(xydata, xlim=NULL, ylim=NULL, eps=1e-09, frac=0.0001)

.expandEdges(polys, pts, xlim, ylim)
```

Arguments

xydata	numeric – data frame with columns X and Y containing the points.
xlim	numeric – range of X-coordinates; a bounding box for the coordinates.
ylim	numeric – range of Y-coordinates; a bounding box for the coordinates.
eps	numeric – the value of epsilon used in testing whether a quantity is zero.
frac	numeric – used to detect duplicate input points, which meet the condition $\lvert x1 - x2 \rvert <$ frac $\times$ (xmax − xmin) and $\lvert y1 - y2 \rvert <$ frac $\times$ (ymax − ymin)
polys	numeric – valid PolySet data frame
pts	numeric – data.frame with 'X' and 'Y' columns

Details

This routine ignores all columns other than 'X' and 'Y'.

If the user leaves 'xlim' and 'ylim' unspecified, the function defaults to the range of the data with each extent expanded by ten percent of the range.

This function sets the attribute 'projection' to 1 and the attribute 'zone' to NULL as it assumes this projection in its calculations.

Value

'PolySet' with columns 'PID', 'POS', 'X', and 'Y'.

Auxiliary dot function '.expandEdges' returns an expanded 'PolySet'.

Author(s)

Nicholas M. Boers, Software Engineer, Jobber, Edmonton AB

Maintainer: Rowan Haigh, Program Head – Offshore Rockfish
Pacific Biological Station (PBS), Fisheries & Oceans Canada (DFO), Nanaimo BC
locus opus: Remote office, Vancouver BC
Last modified Rd: 2024-09-03

See Also

In package **PBSmapping**:
addPoints, addPolys, calcArea, calcCentroid, calcConvexHull, calcMidRange, calcSummary,
locateEvents, plotMap, plotPoints, plotPolys, PolySet

Examples

```
local(envir=.PBSmapEnv,expr={
  oldpar = par(no.readonly=TRUE)
  #--- create some EventData
  events <- as.EventData(data.frame(
    EID=1:200, X=rnorm(200), Y=rnorm(200)), projection=1)
  #--- calculate the Voronoi tesselation
  polys <- calcVoronoi(events)
  #--- create PolyData to color it based on area
  polyData <- calcArea(polys)
  names(polyData)[is.element(names(polyData), "area")] <- "Z"
  colSeq <- seq(0.4, 0.95, length=4)
  polyData <- makeProps(polyData,
    breaks=quantile(polyData$Z,c(0,.25,.5,.75,1)),
    propName="col", propVals=rgb(colSeq,colSeq,colSeq))
  #--- plot the tesselation
  plotMap(polys, polyProps=polyData)
  #--- plot the points
  addPoints(events, pch=19)
  par(oldpar)
})
```

---

clipPolys   *Clip a PolySet as Polygons or Polylines*

Description

Clip a 'PolySet', where each unique ('PID', 'SID') describes a polygon or polyline.

Usage

```
clipPolys (polys, xlim, ylim, keepExtra = FALSE)

clipLines (polys, xlim, ylim, keepExtra = FALSE)

.clip(polys, xlim, ylim, isPolygons, keepExtra)
```

Arguments

`polys`	`data.frame` – `'PolySet'` to clip
`xlim`	`numeric` – range of X-coordinates
`ylim`	`numeric` – range of Y-coordinates
`keepExtra`	`logical` – if TRUE, try to carry forward any non-standard columns into the result
`isPolygons`	`logical` – if TRUE, then `PolySet` contains polygons, else `PolySet` contains polylines

Details

For each discrete polygon, the function connects vertices 1 and N (does not connect vertices 1 and N for discrete polylines). It recalculates the `'POS'` values for each vertex, saving the old values in a column named `'oldPOS'`. For new vertices, it sets `'oldPOS'` to NA.

Value

`'PolySet'` containing the input data, with some points added or removed. A new column `'oldPOS'` records the original `'POS'` value for each vertex.

Author(s)

Nicholas M. Boers, Software Engineer, Jobber, Edmonton AB

Maintainer: Rowan Haigh, Program Head – Offshore Rockfish
Pacific Biological Station (PBS), Fisheries & Oceans Canada (DFO), Nanaimo BC
locus opus: Remote office, Vancouver BC
Last modified Rd: 2024-09-03

See Also

In package **PBSmapping**:
addLines, addPolys, addStipples, PolySet

Examples

```
local(envir=.PBSmapEnv,expr={
  oldpar = par(no.readonly=TRUE)
  #--- create a triangle that will be clipped
  polys <- data.frame(PID=rep(1, 3), POS=1:3, X=c(0,1,.5), Y=c(0,0,1))
  #--- clip the triangle in the X direction, and plot the results
  plotPolys(clipPolys(polys,xlim=c(0,.75),ylim=range(polys[,"Y"])),col=2)
```

```
   par(oldpar)
})
```

---

closePolys   *Close a PolySet*

Description

Close a PolySet of polylines to form polygons.

Usage

```
closePolys (polys)
```

Arguments

polys   PolySet to close.

Details

Generally, run fixBound before this function. The ranges of a PolySet's X and Y columns define the boundary. For each discrete polygon, this function determines if the first and last points lie on a boundary. If both points lie on the same boundary, it adds no points. However, if they lie on different boundaries, it may add one or two corners to the polygon.

When the boundaries are adjacent, one corner will be added as follows:

- top boundary + left boundary implies add top-left corner;
- top boundary + right boundary implies add top-right corner;
- bottom boundary + left boundary implies add bottom-left corner;
- bottom boundary + right boundary implies add bottom-right corner.

When the boundaries are opposite, it first adds the corner closest to a starting or ending polygon vertex. This determines a side (left-right or bottom-top) that connects the opposite boundaries. Then, it adds the other corner of that side to close the polygon.

Value

PolySet identical to polys, except for possible additional corner points.

Author(s)

Nicholas M. Boers, Staff Software Engineer
Jobber, Edmonton AB
Last modified Rd: 2013-04-10

See Also

fixBound, fixPOS.

Examples

```
local(envir=.PBSmapEnv,expr={
  oldpar = par(no.readonly=TRUE)
  #--- 4 corners
  polys <- data.frame(
    PID = c(1, 1, 2, 2, 3, 3, 4, 4),
    POS = c(1, 2, 1, 2, 1, 2, 1, 2),
    X   = c(0, 1, 2, 3, 0, 1, 2, 3),
    Y   = c(1, 0, 0, 1, 2, 3, 3, 2))
  plotPolys(closePolys(polys), col=2)

  #--- 2 corners and 1 opposite
  polys <- data.frame(
    PID = c(1, 1, 2, 2, 3, 3, 3),
    POS = c(1, 2, 1, 2, 1, 2, 3),
    X   = c(0, 1, 0, 1, 5, 6, 1.5),
    Y   = c(1, 0, 2, 3, 0, 1.5, 3))
  plotPolys(closePolys(polys), col=2)
  par(oldpar)
})
```

---

combineEvents   *Combine Measurements of Events*

Description

Combine measurements associated with events that occur in the same polygon.

Usage

```
combineEvents (events, locs, FUN, ..., bdryOK = TRUE)
```

Arguments

events	[EventData](#) with at least four columns (EID, X, Y, Z).
locs	[LocationSet](#) usually resulting from a call to [findPolys](#).
FUN	a function that produces a scalar from a vector (e.g., [mean](#), [sum](#)).
...	optional arguments for FUN.
bdryOK	Boolean value; if TRUE, include boundary points.

Details

This function combines measurements associated with events that occur in the same polygon. Each event (EID) has a corresponding measurement Z. The locs data frame (usually output from [findPolys](#)) places events within polygons. Thus, each polygon (PID, SID) determines a set of events within it, and a corresponding vector of measurements Zv. The function returns FUN(Zv), a summary of measurements within each polygon.

Value

[PolyData](#) with columns PID, SID (*if in* locs), and Z.

Author(s)

[Nicholas M. Boers](#), Staff Software Engineer
Jobber, Edmonton AB
Last modified Rd: 2013-04-10

See Also

[findCells](#), [findPolys](#), [locateEvents](#), [locatePolys](#), [makeGrid](#), [makeProps](#).

Examples

```
local(envir=.PBSmapEnv,expr={
  #--- create an EventData data frame: let each event have Z = 1
  events <- data.frame(EID=1:10, X=1:10, Y=1:10, Z=rep(1, 10))
  #--- example output from findPolys where 1 event occurred in the first
  #--- polygon, 3 in the second, and 6 in the third
  locs <- data.frame(EID=1:10,PID=c(rep(1,1),rep(2,3),rep(3,6)),Bdry=rep(0,10))
  #--- sum the Z column of the events in each polygon, and print the result
  print(combineEvents(events=events, locs=locs, FUN=sum))
})
```

---

combinePolys   *Combine Several Polygons into a Single Polygon*

Description

Combine several polygons into a single polygon by modifying the PID and SID indices.

Usage

```
combinePolys (polys)
```

Arguments

polys   [PolySet](#) with one or more polygons, each with possibly several components/holes.

Details

This function accepts a [PolySet](#) containing one or more polygons (PIDs), each with one or more components or holes (SIDs). The SID column need not exist in the input. The function combines these polygons into a single polygon by simply renumbering the PID and SID indices. The resulting [PolySet](#) contains a single PID (with the value 1) and uses the SID value to differentiate between polygons, their components, and holes.

Value

[PolySet](), possibly with the addition of an SID column if it did not already exist. The function may also reorder columns such that PID, SID, POS, X and Y appear first, in that order.

Author(s)

Nicholas M. Boers, Staff Software Engineer
Jobber, Edmonton AB
Last modified Rd: 2007-06-06

See Also

dividePolys

---

convCP   *Convert Contour Lines into a PolySet*

Description

Convert output from contourLines into a PolySet.

Usage

```
convCP (data, projection = NULL, zone = NULL)
```

Arguments

data	contour line data, often from the contourLines function.
projection	optional projection attribute to add to the PolySet.
zone	optional zone attribute to add to the PolySet.

Details

data contains a list as described below. The contourLines function create a list suitable for the data argument.

A three-element list describes each contour. The named elements in this list include the scalar level, the vector x, and the vector y. Vectors x and y must have equal lengths. A higher-level list (data) contains one or more of these contours lists.

Value

A list with two named elements PolySet and PolyData. The PolySet element contains a PolySet representation of the contour lines. The PolyData element links each contour line (PID, SID) with a level.

Author(s)

[Nicholas M. Boers](), Staff Software Engineer
Jobber, Edmonton AB
Last modified Rd: 2013-04-10

See Also

[contour](), [contourLines](), [convLP](), [makeTopography]().

Examples

```
local(envir=.PBSmapEnv,expr={
  oldpar = par(no.readonly=TRUE)
  #--- create sample data for the contourLines() function
  x <- seq(-0.5, 0.8, length=50);  y <- x
  z <- outer(x, y, FUN = function(x,y) { sin(2*pi*(x^2+y^2))^2; } )
  data <- contourLines(x, y, z, levels=c(0.2, 0.8))
  #--- pass that sample data into convCP()
  result <- convCP(data)
  #--- plot the result
  plotLines(result$PolySet, projection=1)
  print(result$PolyData)
  par(oldpar)
})
```

---

convDP | *Convert EventData/PolyData into a PolySet*

Description

Convert [EventData]()/[PolyData]() into a [PolySet]().

Usage

```
convDP (data, xColumns, yColumns)
```

Arguments

data   [PolyData]() or [EventData]().

xColumns   vector of X-column names.

yColumns   vector of Y-column names.

Details

This function expects data to contain several X- and Y-columns. For example, consider data with columns x1, y1, x2, and y2. Suppose xColumns = c("x1", "x2") and yColumns = c("y1", "y2"). The result will contain nrow(data) polygons. Each one will have two vertices, (x1, y1) and (x2, y2) and POS values 1 and 2, respectively. If data includes an SID column, so will the result.

If data contains an EID and not a PID column, the function uses the EIDs as PIDs.

If data contains both PID and EID columns, the function assumes it is PolyData and ignores the EID column.

Value

PolySet with the same PIDs as those given in data. If data has an SID column, the result will include it.

Author(s)

Nicholas M. Boers, Staff Software Engineer
Jobber, Edmonton AB
Last modified Rd: 2013-04-10

See Also

addPoints, plotPoints.

Examples

```
local(envir=.PBSmapEnv,expr={
  oldpar = par(no.readonly=TRUE)
  #--- create sample PolyData
  polyData <- data.frame(PID=c(1, 2, 3),
    x1=c(1, 3, 5), y1=c(1, 3, 2),
    x2=c(1, 4, 5), y2=c(2, 4, 1),
    x3=c(2, 4, 6), y3=c(2, 3, 1))
  #--- print PolyData
  print(polyData)
  #--- make a PolySet from PolyData
  polys <- convDP(polyData,
    xColumns=c("x1", "x2", "x3"),
    yColumns=c("y1", "y2", "y3"))
  #--- print and plot the PolySet
  print(polys)
  plotLines(polys, xlim=c(0,7), ylim=c(0,5), col=2)
  par(oldpar)
})
```

---

convLP   *Convert Polylines into a Polygon*

Description

Convert two polylines into a polygon.

Usage

```
convLP (polyA, polyB, reverse = TRUE)
```

Arguments

polyA	[PolySet](#) containing a polyline.
polyB	[PolySet](#) containing a polyline.
reverse	Boolean value; if TRUE, reverse polyB's vertices.

Details

The resulting [PolySet](#) contains all the vertices from polyA in their original order. If reverse = TRUE, this function appends the vertices from polyB in the reverse order (nrow(polyB):1). Otherwise, it appends them in their original order. The PID column equals the PID of polyA. No SID column appears in the result. The resulting polygon is an exterior boundary.

Value

[PolySet](#) with a single PID that is the same as polyA. The result contains all the vertices in polyA and polyB. It has the same projection and zone attributes as those in the input PolySets. If an input PolySet's attributes equal NULL, the function uses the other PolySet's. If the PolySet attributes conflict, the result's attribute equals NULL.

Author(s)

[Nicholas M. Boers](#), Staff Software Engineer
Jobber, Edmonton AB
Last modified Rd: 2013-04-10

See Also

[addLines](#), [appendPolys](#), [closePolys](#), [convCP](#), [joinPolys](#), [plotLines](#).

Examples

```
local(envir=.PBSmapEnv,expr={
  oldpar = par(no.readonly=TRUE)
  #--- create two polylines
  polyline1 <- data.frame(PID=rep(1,2),POS=1:2,X=c(1,4),Y=c(1,4))
  polyline2 <- data.frame(PID=rep(1,2),POS=1:2,X=c(2,5),Y=c(1,4))
  #--- create two plots to demonstrate the effect of `reverse'
  par(mfrow=c(2, 1))
  plotPolys(convLP(polyline1, polyline2, reverse=TRUE), col=2)
  plotPolys(convLP(polyline1, polyline2, reverse=FALSE), col=3)
  par(oldpar)
})
```

---

convUL   *Convert Coordinates between UTM and Lon/Lat*

Description

Convert coordinates between UTM and Lon/Lat.

Usage

```
convUL (xydata, km=TRUE, southern=NULL)
```

Arguments

xydata	numeric – data frame with columns X and Y.
km	logical – if TRUE, UTM coordinates within xydata are in kilometres; otherwise, metres.
southern	logical – if TRUE, forces conversions from UTM to longitude/latitude to produce coordinates within the southern hemisphere. For conversions from UTM, this argument defaults to FALSE. For conversions from LL, the function determines southern from xydata.

Details

The object xydata must possess a projection attribute that identifies the current projection. If the data frame contains UTM coordinates, it must also have a zone attribute equal to a number between 1 and 60 (inclusive). If it contains geographic (longitude/latitude) coordinates and the zone attribute is missing, the function computes the mean longitude and uses that value to determine the zone. The longitude range of zone i is $-186 + 6i° < x \le -180 + 6i°$.

This function converts the X and Y columns of xydata from "LL" to "UTM" or vice-versa. If the data span more than **one** zone to the right or left of the intended central zone, the underlying algorithm may produce erroneous results. This limitation means that the user should use the most central zone of the mapped region, or allow the function to determine the central zone when converting from geographic to UTM coordinates. After the conversion, this routine adjusts the data frame's attributes accordingly.

Value

A data frame identical to xydata, except that the X and Y columns contain the results of the conversion, and the projection attribute matches the new projection.

Author(s)

Nicholas M. Boers, Software Engineer, Jobber, Edmonton AB
Maintainer: Rowan Haigh, Program Head – Offshore Rockfish
Pacific Biological Station (PBS), Fisheries & Oceans Canada (DFO), Nanaimo BC
locus opus: Regional Headquarters, Vancouver BC
Last modified Rd: 2022-09-06

References

Ordnance Survey. (2020) A guide to coordinate systems in Great Britain. Copyright Ordnance Survey 2018 **(v3.6)**. Southampton, UK.

See Also

closePolys, fixBound.

Examples

```
local(envir=.PBSmapEnv,expr={
  oldpar = par(no.readonly=TRUE)
  #--- load the data
  data(nepacLL,envir=.PBSmapEnv)
  #--- set the zone attribute
  #--- use a zone that is most central to the mapped region
  attr(nepacLL, "zone") <- 6
  #--- convert and plot the result
  nepacUTM <- convUL(nepacLL)
  plotMap(nepacUTM)
  par(oldpar)
})
```

---

dividePolys   *Divide a Single Polygon into Several Polygons*

Description

Divide a single polygon (with several outer-contour components) into several polygons, a polygon for each outer contour, by modifying the PID and SID indices.

Usage

```
dividePolys (polys)
```

Arguments

polys   [PolySet](#) with one or more polygons, each with possibly several components/holes.

Details

Given the input [PolySet](#), this function renumbers the PID and SID indices so that each outer contour has a unique PID and is followed by all of its holes, identifying them with SIDs greater than one.

Value

[PolySet](#), possibly with the addition of an SID column if it did not already exist. The function may also reorder columns such that PID, SID, POS, X and Y appear first, in that order.

Author(s)

[Nicholas M. Boers](#), Staff Software Engineer
Jobber, Edmonton AB
Last modified Rd: 2007-06-06

See Also

[combinePolys](#).

---

dot-add   *Add Items to Plots*

Description

Routines to add various items to existing plots.

Usage

```
.addAxis(xlim, ylim, tckLab, tck, tckMinor, ...)
.addAxis2(side=1:2, xlim, ylim, tckLab, tck, tckMinor, ...)
.addBubblesLegend(radii.leg, usr.xdiff, usr.ydiff, symbol.zero, symbol.fg, symbol.bg,
   legend.pos, legend.breaks, legend.type, legend.title, legend.cex, ...)
.addCorners(polys, ptSummary)
.addFeature(feature, data, polyProps, isEventData, cex=NULL, col=NULL, font=NULL,
   pch=NULL, ...)
.addLabels(projection=NULL, ...)
.addProps(type, polyProps, ...)
```

Arguments

`xlim`	numeric – limits of the x-axis
`ylim`	numeric – limits of the y-axis
`tckLab`	character – tick labels
`tck`	numeric – length of tick marks as a fraction of the smaller of the width or height of the plotting region
`tckMinor`	numeric – length of minor tick marks between major tick marks
`...`	dots – additional arguments used by functions within function
`side`	numeric – side of the plot (1=bottom, 2=left, 3=top, 4=right)
`radii.leg`	numeric – radii of bubbles in legend
`usr.xdiff`	numeric – difference between the limits of x
`usr.ydiff`	numeric – difference between the limits of y
`symbol.zero`	character – character to use for plotting zero values
`symbol.fg`	numeric – colour of bubble outline
`symbol.bg`	numeric – colour of bubble fill
`legend.pos`	character – one of legends positional strings (e.g., 'bottomleft')
`legend.breaks`	numeric – break values for categorizing z-values
`legend.type`	character – display format for legend
`legend.title`	character – title for legend
`legend.cex`	numeric – size of legend text
`polys`	PolySet – polygon data file
`ptSummary`	numeric – used in computation of distance from candidate points to corner
`feature`	character – if `'points'`, add `'data'` using `'points()'` if `'labels'`, add `'data'` using `'text()'`
`data`	data.frame – one of the data structures used in `'PBSmapping'`
`polyProps`	data.frame – polygon properties, like a `'PolyData'` object
`isEventData`	logical – if TRUE, look for `'EID'` column in `'data'`
`cex`	numeric – value to magnify text and symbols relative to the default
`col`	character|numeric – colour for added feature
`font`	numeric – font number for added feature
`pch`	numeric – plotting symbol for added feature
`projection`	character – projection of plot on which to add label (either `'LL'` or `'UTM'`)
`type`	character – data structure type, where `'e'` = `'EventData'` and `'p'` = `'PolyData'`

Details

Internal functions add features to an existing map plot.

Value

Nothing in particular.

Author(s)

Nicholas M. Boers, Software Engineer, Jobber, Edmonton AB

Maintainer: Rowan Haigh, Program Head – Offshore Rockfish
Pacific Biological Station (PBS), Fisheries & Oceans Canada (DFO), Nanaimo BC
locus opus: Remote office, Vancouver BC
Last modified `Rd: 2024-09-03`

See Also

In package **PBSmapping**:
`plotMap`, `addBubbles`, `addPolys`, `addLabels`, `addStipples`

---

dot-calc   *Calculate Things*

Description

Calculate distance and/or orientation.

Usage

```
.calcDist(polys)
.calcOrientation(polys)
```

Arguments

polys   `data.frame` – valid `PolySet` with `'projection'` attribute

Details

`.calcdist` : Equatorial radius 6,378.14 km; Polar radius 6,356.78 km; Mean radius 6,371.3 km

`.calcOrientation` : Calls C code `'calcOrientation'` using `.C()`.

Value

`.calcdist` : distance vector (distances between each point)

`.calcOrientation` : data frame with `'orientation'` column (-1 when counter-clockwise; 0 when N/A; +1 when clockwise) or NULL if no rows in output

Author(s)

Nicholas M. Boers, Software Engineer, Jobber, Edmonton AB

Maintainer: Rowan Haigh, Program Head – Offshore Rockfish
Pacific Biological Station (PBS), Fisheries & Oceans Canada (DFO), Nanaimo BC
locus opus: Remote office, Vancouver BC
Last modified Rd: 2024-09-03

See Also

In package **PBSmapping**:
calcLength, placeHoles, thickenPolys

---

dot-check   *Check Stuff*

Description

Auxiliary (dot) functions to check various attributes of PBmapping objects.

Usage

```
.checkClipLimits(limits)
.checkProjection(projectionPlot, projectionPoly)
.checkRDeps(caller="unspecified", requires=NULL)
```

Arguments

limits	numeric – 4-element vector of limits for 'X' and 'Y'
projectionPlot	character – projection of the currently plotted map
projectionPoly	character – projection of the PolySet, presumably to add to map
caller	character – a function name
requires	character – a vector of package names

Details

Auxiliary dot functions to facilitate the machinations of PBSmapping.

Value

.checkClipLimits : nada; just a limit checker
.checkProjection : nada; just a projection checker
.checkRDeps : nada; just a package checker

Author(s)

Nicholas M. Boers, Software Engineer, Jobber, Edmonton AB

Maintainer: Rowan Haigh, Program Head – Offshore Rockfish
Pacific Biological Station (PBS), Fisheries & Oceans Canada (DFO), Nanaimo BC
locus opus: Remote office, Vancouver BC
Last modified Rd: 2024-09-03

See Also

In package **PBSmapping**:
.checkClipLimits used by:
importGSHHS
.checkProjection used by:
addLabels, addLines, addPoints, addPolys, addStipples
.checkRDeps used by:
calcVoronoi, importShapefile (temporarily unavailable)

---

dot-create   *Create Indices for PolySets*

Description

Create indices for indexing map data structures (PolySets).

Usage

```
.createIDs(x, cols, fastIDdig=NULL)
.createFastIDdig(polysA, polysB=NULL, cols)
.createGridIDs(d, addSID, byrow)
```

Arguments

x	data.frame – PolySet with one or more ID columns
cols	character – field (or column) names (e.g., ('PID','SID'))
fastIDdig	numeric – maximum number of digits in the second column; often the output from '.createFastIDdig' is passed to 'fastIDdig' when creating (some) matching indices for two different data frames
polysA	data.frame – first PolySet data frame
polysB	data.frame – second PolySet data frame, which may be missing one or both 'cols'
d	data.frame – grid Polyset created using addSID=T and byrow=T
addSID	logical – if TRUE, include a 'SID' column
byrow	logical – if TRUE, increment 'PID' along 'X'

Details

`.createIDs` : create IDs (or IDX) column from its input.

`.createFastIDdig` : determine the maximum number of digits in the second column of a data frame. If given two data frames (`'polysA'` and `'polysB'`), determine the maximum between the two data frames.

`.createGridIDs` : Create IDs for a grid according to the `addSID` and `byrow` arguments.

Value

`.createIDs` : a vector of integer or real-number indices.

`.createFastIDdig` : maximum number of digits to use in real-number index.

`.createGridIDs` : a modified grid PolySet.

Author(s)

Nicholas M. Boers, Software Engineer, Jobber, Edmonton AB

Maintainer: Rowan Haigh, Program Head – Offshore Rockfish
Pacific Biological Station (PBS), Fisheries & Oceans Canada (DFO), Nanaimo BC
locus opus: Remote office, Vancouver BC
Last modified Rd: 2024-09-25

See Also

In package **PBSmapping**:
[plotMap](), [addBubbles](), [addPolys](), [addLabels](), [addStipples]()

---

dot-fix   *Fix Things*

Description

Fix things like GSHHSWorld.

Usage

```
.fixGSHHSWorld(world)
```

Arguments

world   data.frame – PolySet of polygons for the global shorelines

Details

Determine PID of Antarctica, which is used to extract the current Antarctica. The continent is rebuilt from left to right, creating a new, very wide Antarctica. It is then clipped and merged into the existing world (with other polygons).

Value

A revised `PolySet` of the world with a peeled version of Antarctica.

Note

This function is not called by any PBSmapping functions. It can be used after `'importGSHHS'` uses `ylim=c(-90,90)`.

Author(s)

Nicholas M. Boers, Software Engineer, Jobber, Edmonton AB

Maintainer: Rowan Haigh, Program Head – Offshore Rockfish
Pacific Biological Station (PBS), Fisheries & Oceans Canada (DFO), Nanaimo BC
locus opus: Remote office, Vancouver BC
Last modified Rd: 2024-09-03

See Also

In package **PBSmapping**:
`importGSHHS`, `clipPolys`

Examples

```
## Not run:
worldLL =
importGSHHS("gshhs_l.b",xlim=c(-20,360),ylim=c(-90,90),level=1,n=15,xoff=0)
worldLL = .fixGSHHSWorld(worldLL)

## End(Not run)
```

---

dot-get   *Get Stuff*

Description

Routines to get various attributes from `PBmapping` objects.

Usage

```
.getBasename(fn, ext)
.getGridPars(polys, fullValidation=TRUE)
```

Arguments

fn	character – file name (primarily used for shapefiles), with or without extension
ext	character – extension text (perhaps only applicable to Windows OS)
polys	numeric – PolySet data frame
fullValidation	logical – if TRUE, apply the full validation routine on the PolySet

Details

Auxiliary dot functions to facilitate the machinations of `PBSmapping`.

Value

`.getBasename` : base name of a shapefile set (usually with multiple extensions).
`.getGridPars` : PBSmapping grid object attributes.

Author(s)

Nicholas M. Boers, Software Engineer, Jobber, Edmonton AB

Maintainer: Rowan Haigh, Program Head – Offshore Rockfish
Pacific Biological Station (PBS), Fisheries & Oceans Canada (DFO), Nanaimo BC
locus opus: Remote office, Vancouver BC
Last modified Rd: 2024-09-03

See Also

In package **PBSmapping**:
importShapefile (temporarily unavailable), `makeGrid`, `is.PolySet`

---

dot-is.in *Are Points in Polygons?*

Description

Determines which points (EventData) are located inside a polygon (PolySet).

Usage

```
.is.in(events, polys, use.names=TRUE)
```

Arguments

events	numeric – data frame of event data (points), with X and Y coordinates; not necessarily a PBSmapping `EventData` object
polys	numeric – a PBSmapping PolySet (polygon)
use.names	logical – *currently not used*

Details

Taps into the PBSmapping C code for `'findPolys'`.

Value

Reports events inside the polygon, outside the polygon, and whether all events are inside, outside, or on the boundary of the polygon.

Author(s)

[Rowan Haigh](#), Program Head – Offshore Rockfish
Pacific Biological Station (PBS), Fisheries & Oceans Canada (DFO), Nanaimo BC
locus opus: Remote office, Vancouver BC
Last modified `Rd: 2024-03-11`

See Also

In **PBSmapping**:
[findPolys](#), [as.EventData](#), [is.PolySet](#), [findPolys](#)

---

dot-mat2df   *Convert Matrix to Data Frame*

Description

Convert PBS matrices to data frames, preserving certain attributes.

Usage

```
.mat2df(data)
```

Arguments

data   `numeric` – matrix, usually numeric but could be all character (mixing data
                modes usually not allowed in matrices/arrays)

Details

Converts a PBS matrix object to a PBS data frame. Preserves PBS class objects `'EventData'`,
`'LocationSet'`, `'PolyData'`, `'PolySet'`, if any of them exist in the matrix.

Value

Transformed data object (from matrix to data frame).

Author(s)

[Nicholas M. Boers](#), Software Engineer, Jobber, Edmonton AB

Maintainer: [Rowan Haigh](#), Program Head – Offshore Rockfish
Pacific Biological Station (PBS), Fisheries & Oceans Canada (DFO), Nanaimo BC
locus opus: Remote office, Vancouver BC
Last modified `Rd: 2024-09-03`

See Also

In package **PBSmapping**:
[addPoints](#), [.addFeature](#), [.validateData](#)

---

dot-PBSfigs   *Run PBSmapping Example Figures*

Description

Run built-in example figures for PBSmapping.

Usage

```
.PBSfigs(nfigs=1:10, wait=TRUE)
.PBSfig01()
.PBSfig02()
.PBSfig03()
.PBSfig04()
.PBSfig05()
.PBSfig06()
.PBSfig07()
.PBSfig08()
.PBSfig09()
.PBSfig10()
```

Arguments

nfigs	numeric – figure number(s) to run
wait	logical – if TRUE, figure pauses until user hits <return> for the next figure to display.

Details

The wrapper function `'.PBSfigs'` displays pre-defined figures from 1 to 10, depending on the numbers that the user specifies. User can also call each figure directly, e.g., `'.PBSfig05()'`.

Value

NULL

Author(s)

Nicholas M. Boers, Software Engineer, Jobber, Edmonton AB
Maintainer: Rowan Haigh, Program Head – Offshore Rockfish
Pacific Biological Station (PBS), Fisheries & Oceans Canada (DFO), Nanaimo BC
locus opus: Offsite, Vancouver BC
Last modified Rd: 2023-10-30

Examples

```
.PBSfigs(nfigs=1:3, wait=FALSE)
```

---

dot-PBSmapEnv   *PBSmapping Environment*

Description

An environment set aside for PBSmapping.

Usage

```
.PBSmapEnv
```

Format

A new environment with a `.GlobalEnv` parent.

Details

The environment is created in `'zzz.r'` and can be used by PBSmodelling functions `'lisp'`, `'tget'`, `'tput'`, `'tprint'`, and `'tcall'`, if this package has be loaded.

Currently, `'.PBSmapEnv'` is only used in `'.validateData'` (in the namespace), and by a set of print functions specific to PBSmapping.

Source

Generated by a call to the base function `new.env()`.

See Also

In **PBSmapping**:
[print.EventData](), [print.LocationSet](), [print.PolyData](), [print.PolySet](), [print.summary.PBS]()
In **PBSmodelling**:
lisp, tget, packList

---

dot-validate   *Validate PBSmapping Data Structures*

Description

Routines to validate data structures (e.g., required field names, attributes) that are passed to package functions.

Usage

```
.validateData(data, className, requiredCols=NULL, requiredAttr=NULL,
    noFactorCols=NULL, noNACols=NULL, keyCols=NULL, numericCols=NULL)
.validateEventData(EventData)
.validateLocationSet(LocationSet)
.validatePolyData(PolyData)
.validatePolyProps(polyProps, parCols=NULL)
.validatePolySet(polys)
.validateXYData(xyData)
```

Arguments

data	`data.frame|matrix` – data object for use in **PBSmapping** functions
className	`character` – class name of data, usually `'data.frame'`
requiredCols	`character` – vector of required field names in data
requiredAttr	`character` – names of attributes to check
noFactorCols	`character` – vector of field names that cannot be factors
noNACols	`character` – vector of field names that cannot comtain NA values
keyCols	`character` – vector of field names that must contain unique values
numericCols	`character` – vector of field names that must be numeric
EventData	`data.frame` – data frame with at least three fields named (`'EID'`, `'X'`, `'Y'`)
LocationSet	`data.frame` – data frame with three or four columns (`'EID'`, `'PID'`, `'SID'`, `'Bdry'`), where `'SID'` may be missing
PolyData	`data.frame` – data frame with a first column named `'PID'` and (optionally) a second column named `'SID'`
polyProps	`data.frame` – support PolyData object, usually containing fields that facilitate plotting, either derived from a PolySet summary or given as an explicit PolyData object
parCols	`character` – vector of field names that cannot be factors
polys	`data.frame` – a collection of polygonal contours (PolySet) based on four or five numerical fields: `'PID'` the primary identification number for a contour `'SID'` (optional) the secondary identification number for a contour `'POS'` the position number associated with a vertex `'X'` the horizontal coordinate at a vertex `'Y'` the vertical coordinate at a vertex
xyData	`data.frame` – a collection of events that contain fields `'X'` and `'Y'`

Details

Internal functions check the validity of data objects used by **PBSmapping**. The primary function is `'.validateData'`; other dot `validate` functions are wrappers for the four main data structures: `'PolySet'`, `'PolyData'`, `'EventData'`, and `'LocationSet'`.

Value

The data object that is validated.

Author(s)

Nicholas M. Boers, Software Engineer, Jobber, Edmonton AB

Maintainer: Rowan Haigh, Program Head – Offshore Rockfish
Pacific Biological Station (PBS), Fisheries & Oceans Canada (DFO), Nanaimo BC
locus opus: Remote office, Vancouver BC
Last modified `Rd: 2024-09-03`

See Also

In package **PBSmapping**:
`PolySet`, `PolyData`, `EventData`, `LocationSet`

EventData   *EventData Objects*

Description

An EventData object comprises a data frame with at least three fields named `EID`, `X`, and `Y`; each row specifies an event that occurs at a specific point.

PBSmapping functions that expect EventData will accept properly formatted data frames in their place (see 'Details').

`as.EventData` attempts to coerce a data frame to an object with class EventData.

`is.EventData` returns `TRUE` if its argument is of class EventData.

Usage

```
as.EventData(x, projection = NULL, zone = NULL)
is.EventData(x, fullValidation = TRUE)
```

Arguments

x	data frame to be coerced or tested.
projection	optional `projection` attribute to add to EventData, possibly overwriting an existing attribute.
zone	optional `zone` attribute to add to EventData, possibly overwriting an existing attribute.
fullValidation	Boolean value; if `TRUE`, fully test x.

Details

Conceptually, an EventData object describes events (EID) that take place at specific points (X,Y) in two-dimensional space. Additional fields can specify measurements associated with these events. In a fishery context, EventData could describe fishing events associated with trawl tows, based on the fields:

- `EID` - fishing event (tow) identification number;
- `X, Y` - fishing location;
- `Duration` - length of time for the tow;
- `Depth` - average depth of the tow;
- `Catch` - biomass captured.

Like [PolyData](#), EventData can have attributes `projection` and `zone`, which may be absent. Inserting the string `"EventData"` as the class attribute's first element alters the behaviour of some functions, including [print](#) (if [PBSprint](#) is TRUE) and [summary](#).

Value

The `as.EventData` method returns an object with classes `"EventData"` and `"data.frame"`, in that order.

Author(s)

[Nicholas M. Boers](#), Staff Software Engineer
Jobber, Edmonton AB
Last modified Rd: 2015-04-23

See Also

[PolySet](#), [PolyData](#), [LocationSet](#)

---

extractPolyData   *Extract PolyData from a PolySet*

Description

Extract [PolyData](#) from a [PolySet](#). Columns for the [PolyData](#) include those other than PID, SID, POS, oldPOS, X, and Y.

Usage

```
extractPolyData (polys)
```

Arguments

polys   [PolySet](#) to use.

Details

This function identifies the [PolySet](#)'s extra columns and determines if those columns contain unique values for each (PID, SID). Where they do, the (PID, SID) will appear in the [PolyData](#) output with that unique value. Where they do not, the extra column will contain NAs for that (PID, SID).

Value

[PolyData](#) with columns PID, SID, and any extra columns.

Author(s)

[Nicholas M. Boers](#), Staff Software Engineer
Jobber, Edmonton AB
Last modified Rd: 2013-04-10

See Also

[makeProps](#), [PolyData](#), [PolySet](#).

Examples

```
local(envir=.PBSmapEnv,expr={
  #--- create a PolySet with an extra column
  polys <- data.frame(PID = c(rep(1, 10), rep(2, 10)),
    POS = c(1:10, 1:10),
    X = c(rep(1, 10), rep(1, 10)),
    Y = c(rep(1, 10), rep(1, 10)),
    colour = (c(rep("green", 10), rep("red", 10))))
  #--- extract the PolyData
  print(extractPolyData(polys))
})
```

---

findCells   *Find Grid Cells that Contain Events*

Description

Find the grid cells in a [PolySet](#) that contain events specified in [EventData](#). Similar to [findPolys](#), except this function requires a [PolySet](#) resulting from [makeGrid](#). This restriction allows this function to calculate the result with greater efficiency.

Usage

```
findCells (events, polys, includeBdry=NULL)
```

Arguments

events	[EventData](#) to use.
polys	[PolySet](#) to use.
includeBdry	numeric: determines how points on boundaries are handled:
	if NULL then report all points on polygon boundaries (default behaviour);
	if 0 then exclude all points on polygon boundaries;
	if 1 then report only the first (lowest PID/SID) polygon boundary;
	if 2,...,n then report the last (highest PID/SID) polygon boundary.

Details

The resulting data frame, a [LocationSet](#), contains the columns EID, PID, SID (*if in* polys), and Bdry, where an event (EID) occurs in a polygon (PID, SID). The Boolean (0,1) variable Bdry indicates whether an event lies on a polygon's edge. Note that if an event lies properly outside of all the polygons, then a record with (EID, PID, SID) does not occur in the output. It may happen, however, that an event occurs in multiple polygons (i.e., on two or more boundaries). Thus, the same EID can occur more than once in the output.

If an event happens to lie at the boundary intersection of four (or two) grid cells then one EID will be associated with four (or two) grid cells. A user can choose to manipulate this result by setting the argument includeBdry to a numeric value that constrains the association of a boundary event to 0 or 1 grid cell (see argument description above).

Value

[LocationSet](#) that links events with polygons.

Author(s)

[Nicholas M. Boers](#), Staff Software Engineer
Jobber, Edmonton AB
Last modified Rd: 2014-12-15

See Also

[findPolys](#), [makeGrid](#), [combineEvents](#), [locateEvents](#), [locatePolys](#), [LocationSet](#).

Examples

```
local(envir=.PBSmapEnv,expr={
  oldpar = par(no.readonly=TRUE)
  #--- create some EventData: points in a diagonal line
  events <- data.frame(EID=1:11, X=seq(0, 2, length=11),
    Y=seq(0, 2, length=11))
  events <- as.EventData(events, projection=1);
  #--- create a PolySet (a grid)
  polys <- makeGrid (x=seq(0, 2, by=0.50), y=seq(0, 2, by=0.50), projection=1)
  #--- show a picture
  plotPolys(polys, xlim=range(polys$X)+c(-0.1, 0.1),
    ylim=range(polys$Y)+c(-0.1, 0.1), projection=1)
```

```
   addPoints(events, col=2)
   #--- run findCells and print the results
   fc <- findCells(events, polys)
   fc <- fc[order(fc$EID, fc$PID, fc$SID), ]
   fc$label <- paste(fc$PID, fc$SID, sep=", ")
   print (fc)
   #--- add labels to the graph
   addLabels(as.PolyData(fc[!duplicated(paste(fc$PID,fc$SID)), ],
     projection=1), placement="CENTROID",
     polys=as.PolySet(polys, projection=1), col=4)
   par(oldpar)
})
```

---

findPolys   *Find Polygons that Contain Events*

Description

Find the polygons in a [PolySet](#) that contain events specified in [EventData](#).

Usage

```
findPolys (events, polys, maxRows = 1e+05, includeBdry=NULL)
```

Arguments

events	[EventData](#) to use.
polys	[PolySet](#) to use.
maxRows	estimated maximum number of rows in the output [LocationSet](#).
includeBdry	numeric: determines how points on boundaries are handled: if NULL then report all points on polygon boundaries (default behaviour); if 0 then exclude all points on polygon boundaries; if 1 then report only the first (lowest PID/SID) polygon boundary; if 2, . . . ,n then report the last (highest PID/SID) polygon boundary.

Details

The resulting data frame, a [LocationSet](#), contains the columns EID, PID, SID (*if in* polys), and Bdry, where an event (EID) occurs in a polygon (PID, SID) and SID does not correspond to an inner boundary. The Boolean variable Bdry indicates whether an event lies on a polygon's edge. Note that if an event lies properly outside of all the polygons, then a record with (EID, PID, SID) does not occur in the output. It may happen, however, that an event occurs in multiple polygons. Thus, the same EID can occur more than once in the output.

If an event happens to lie at the boundary intersection of two or more polygons then one EID will be associated with two or more polygons. A user can choose to manipulate this result by setting the argument includeBdry to a numeric value that constrains the association of a boundary event to 0 or 1 polygon (see argument description above).

Value

[LocationSet](#) that links events with polygons.

Author(s)

[Nicholas M. Boers](#), Staff Software Engineer
Jobber, Edmonton AB
Last modified `Rd: 2014-12-15`

See Also

[combineEvents](#), [findCells](#), [locateEvents](#), [locatePolys](#), [LocationSet](#), [makeGrid](#).

Examples

```
local(envir=.PBSmapEnv,expr={
  oldpar = par(no.readonly=TRUE)
  #--- create some EventData: a column of points at X = 0.5
  events <- data.frame(EID=1:10, X=.5, Y=seq(0, 2, length=10))
  events <- as.EventData(events, projection=1)
  #--- create a PolySet: two squares with the second above the first
  polys <- data.frame(PID=c(rep(1, 4), rep(2, 4)), POS=c(1:4, 1:4),
    X=c(0, 1, 1, 0, 0, 1, 1, 0),
    Y=c(0, 0, 1, 1, 1, 1, 2, 2))
  polys <- as.PolySet(polys, projection=1)
  #--- show a picture
  plotPolys(polys, xlim=range(polys$X)+c(-0.1, 0.1),
    ylim=range(polys$Y)+c(-0.1, 0.1), projection=1);
  addPoints(events, col=2);
  #--- run findPolys and print the results
  print(findPolys(events, polys))
  par(oldpar)
})
```

---

fixBound   *Fix the Boundary Points of a PolySet*

Description

The ranges of a [PolySet](#)'s X and Y columns define its boundary. This function fixes a [PolySet](#)'s vertices by moving vertices near a boundary to the actual boundary.

Usage

```
fixBound (polys, tol)
```

Arguments

polys   [PolySet](#) to fix.

tol   vector (length 1 or 2) specifying a percentage of the ranges to use in defining
               near to a boundary. If tol has two elements, the first specifies the tolerance for
               the x-axis and the second the y-axis. If it has only one element, the function uses
               the same tolerance for both axes.

Details

When moving vertices to a boundary, the function moves them strictly horizontally or vertically, as
appropriate.

Value

[PolySet](#) identical to the input, except for possible changes in the X and Y columns.

Author(s)

[Nicholas M. Boers](#), Staff Software Engineer
Jobber, Edmonton AB
Last modified Rd: 2013-04-10

See Also

[closePolys](#), [fixPOS](#), [isConvex](#), [isIntersecting](#), [PolySet](#).

Examples

```
local(envir=.PBSmapEnv,expr={
  oldpar = par(no.readonly=TRUE)
  #--- set up a long horizontal and long vertical line to extend the plot's
  #--- limits, and then try fixing the bounds of a line in the top-left
  #--- corner and a line in the bottom-right corner
  polys <- data.frame(PID=c(1, 1, 2, 2,  3, 3, 4, 4),
    POS=c(1, 2, 1, 2,  1, 2, 1, 2),
    X = c(0, 10, 5, 5, 0.1, 4.9, 5.1, 9.9),
    Y = c(5, 5, 0, 10, 5.1, 9.9, 0.1, 4.9))
  polys <- fixBound(polys, tol=0.0100001)
  plotLines(polys)
  par(oldpar)
})
```

---

fixPOS *Fix the POS Column of a PolySet*

Description

Fix the POS column of a [PolySet](#) by recalculating it using sequential integers.

Usage

```
fixPOS (polys, exteriorCCW = NA)
```

Arguments

polys   [PolySet](#) to fix.

exteriorCCW   Boolean value; if TRUE, orders exterior polygon vertices in a counter-clockwise
               direction. If FALSE, orders them in a clockwise direction. If NA, maintains their
               original order.

Details

This function recalculates the POS values of each (PID, SID) as either 1 to N or N to 1, depending on
the order of POS (ascending or descending) in the input data. POS values in the input must be properly
ordered (ascending or descending), but they may contain fractional values. For example, POS =
2.5 might correspond to a point manually added between POS = 2 and POS = 3. If exteriorCCW =
NA, all other columns remain unchanged. Otherwise, it orders the X and Y columns according to
exteriorCCW.

Value

[PolySet](#) with the same columns as the input, except for possible changes to the POS, X, and Y
columns.

Author(s)

[Nicholas M. Boers](#), Staff Software Engineer
Jobber, Edmonton AB
Last modified Rd: 2013-04-10

See Also

[closePolys](#), [fixBound](#), [isConvex](#), [isIntersecting](#), [PolySet](#).

Examples

```
local(envir=.PBSmapEnv,expr={
  #--- create a PolySet with broken POS numbering
  polys <- data.frame(PID = c(rep(1, 10), rep(2, 10)),
    POS = c(seq(2, 10, length = 10), seq(10, 2, length = 10)),
    X = c(rep(1, 10), rep(1, 10)),
    Y = c(rep(1, 10), rep(1, 10)))
  #--- fix the POS numbering
  polys <- fixPOS(polys)
  #--- print the results
  print(polys)
})
```

---

importEvents   *Import EventData from a Text File*

Description

Import a text file and convert into `EventData`.

Usage

```
importEvents(EventData, projection=NULL, zone=NULL)
```

Arguments

EventData	character – filename of `EventData` text file.
projection	character – optional projection attribute to add to `EventData`.
zone	numeric – optional zone attribute to add to `EventData`.

Value

An imported `EventData`.

Author(s)

[Nicholas M. Boers](#), Staff Software Engineer
Jobber, Edmonton AB
Last modified Rd: 2023-10-30

See Also

[importPolys](#), [importLocs](#), [importGSHHS](#)

---

importGSHHS   *Import Data from a GSHHS Database*

Description

Import data from a GSHHS database and convert data into a `PolySet` with a `PolyData` attribute. The database was originally called 'Global Self-consistent, Hierarchical, High-resolution Shoreline' (GSHHS, Wessel and Smith 1996), but 'Shoreline' was subsequently expanded to include more 'Geography' (GSHHG).

Usage

```
importGSHHS(gshhsDB, xlim, ylim, maxLevel=4, n=0, useWest=FALSE)
```

Arguments

gshhsDB	character – path name to binary GSHHS database. If unspecified, looks for gshhs_f.b in the root of the **PBSmapping** library directory.
xlim	numeric – range of X-coordinates (for clipping). The range should be between 0 and 360, starting at the Greenwich meridian and wrapping eastward around the globe.
ylim	numeric – range of Y-coordinates (for clipping).
maxLevel	numeric – maximum level of polygons to import: 1 (land), 2 (lakes on land), 3 (islands in lakes), or 4 (ponds on islands); ignored when importing lines.
n	numeric – minimum number of vertices that must exist in a line/polygon in order for it to be imported.
useWest	logical – if TRUE, convert the X-coordinates (longitude) to °W (western hemisphere -180 to 0, i.e., west of the Greenwich meridian).

Details

This routine requires a binary GSHHG (Global Self-consistent, Hierarchical, High-resolution Geography) database file. The GSHHG database has been released in the public domain. (www.soest.hawaii.edu/pwessel/gshhg)
At the time of writing, the most recent binary database was the archive file called gshhg-bin-2.3.7.zip.

The archive contains multiple binary files that contain geographical coordinates for shorelines (gshhs), rivers (wdb_rivers), and borders (wdb_borders). The latter two come from World Data-Bank II (WDBII). The five resolutions available are: full (f), high (h), intermediate (i), low (l), and coarse (c).

This routine returns a `PolySet` object with an associated `PolyData` attribute. The attribute contains four fields: (a) `PID`, (b) `SID`, (c) `Level`, and (d) `Source`. Each record corresponds to a line/polygon in the `PolySet`. The `Level` indicates the line's/polygon's level (1=land, 2=lake, 3=island, 4=pond). The `Source` identifies the data source (1=WVS, 0=CIA (WDBII)).

Value

A `PolySet` with a `PolyData` attribute.

Note

The function calls a C routine, also called `importGSHHS`, which returns a set of map coordinates that is not always predictably laid out. This issue stems from how the world is divided at the Greenwich meridian and at the International Date Line. The unpredictability occurs when user-specified X-limits span either of the longitudinal meridians – $(0°, 360°)$ or $(-180°, 180°)$.

This version of the R function attempts to stitch together the overlapping edges of `gshhs` that run from $-20°$ to $360°$ (see example map 5 below). At present, no attempt has been made to deal with the overlap at the International Date Line where Russia overlaps the Aleutian Islands of Alaska. To some extent, the C-code can deal with this, but not in all cases.

Therefore, the user will likely experience some limitations when using `importGSHHS`. The solution is to import the whole dataset with this function using `xlim=c(0,360)`, and then apply the function `refocusWorld` with user-desired X-limits. The Y-limits are generally not problematic unless the user wants to focus on either pole.

Author(s)

Nicholas M. Boers, Senior Software Engineer
Jobber, Edmonton AB

Maintainer: Rowan Haigh, Program Head – Offshore Rockfish
Pacific Biological Station (PBS), Fisheries & Oceans Canada (DFO), Nanaimo BC
locus opus: Offsite, Vancouver BC
Last modified `Rd`: `2024-09-25`

References

Wessel, P., and Smith, W.H.F. (1996) A Global Self-consistent, Hierarchical, High-resolution Shoreline Database. *J. Geophys. Res.* **101** 8741-8743.
(www.soest.hawaii.edu/pwessel/gshhg/Wessel+Smith_1996_JGR.pdf)

See Also

In package **PBSmapping**:
`importEvents`, `importLocs`, `importPolys`

Examples

```
## Not run:
useWest=FALSE
useVers=c("2.2.0","2.2.3","2.3.0","2.3.4") # GSHHG versions
mapswitch = 5
for (i in c("land","rivers","borders"))
  if (exists(i)) eval(parse(text=paste0("rm(",i,")")))
switch( mapswitch,
 # 1. Canada---------------------------------------------
     {vN=4; useWest=T; xlim=c(-150,-50)+360;ylim=c(40,75)},
```

```
   # 2. NW Canada & America----------------------------------
        {vN=4; useWest=T;xlim=c(-136,-100)+360;ylim=c(40,75)},
   # 3. Black Sea (user Ivailo)------------------------------
        {vN=4; xlim=c(27.5, 34.3); ylim=c(40.9, 46.7)},
   # 4. W Europe, NW Africa (user Uli)----------------------
        {vN=4; xlim=c(-20,10); ylim=c(20,50)},
   # 5. W Europe + Iceland------------------------------------
        {vN=4; xlim=c(-25, 20); ylim=c(40, 68)},
   # 6. New Zealand-------------------------------------------
        {vN=4; xlim=c(163, 182); ylim=c(-48,-34)},
   # 7. Australia---------------------------------------------
        {vN=4; xlim=c(112,155); ylim=c(-44,-10)},
   # 8. Japan-------------------------------------------------
        {vN=4; xlim=c(127,148); ylim=c(30,47)},
   # 9. Central America---------------------------------------
        {vN=4; useWest=T; xlim=c(-95,-60)+360;ylim=c(-10,25)},
   #10. North Pacific----------------------------------------
        {vN=4; useWest=T; xlim=c(150,220); ylim=c(45,80)},
   #11. Pacific Ocean----------------------------------------
        {vN=4; xlim=c(112,240); ylim=c(-48,80)},
   #12. North Atlantic (world coordinates)-------------------
        {vN=4; xlim=c(285,360); ylim=c(40,68)},
   #13. North Atlantic (western hemisphere coordinates)-------
        {vN=4; xlim=c(-75,0); ylim=c(40,68)},
   #14. Atlantic Ocean---------------------------------------
        {vN=4; xlim=c(285,380); ylim=c(-50,68)},
   #15. Northern hemisphere----------------------------------
        {vN=4; xlim=c(-180,180); ylim=c(0,85)},
   #16. Asia-------------------------------------------------
        {vN=4; xlim=c(0,180); ylim=c(0,80)},
   #17. North America----------------------------------------
        {vN=4; xlim=c(-180,0); ylim=c(0,80)},
   #18. International date line------------------------------
        {vN=4; xlim=c(45,315); ylim=c(0,80)},
   #19. Indian Ocean-----------------------------------------
        {vN=4; xlim=c(20,130); ylim=c(-40,40)},
   #20. Moose County ("400 miles north of everywhere")-------
        {vN=4; xlim=c(272.5,280.5); ylim=c(43,47.5)}
)
db=paste0("gshhg-bin-",useVers[vN])        # database version folder
gshhg  = paste0("C:/Ruser/GSHHG/",db,"/") # directory with binary files
land   = importGSHHS(paste0(gshhg,"gshhs_i.b"),
           xlim=xlim,ylim=ylim,maxLevel=4,useWest=useWest)
rivers = importGSHHS(paste0(gshhg,"wdb_rivers_i.b"),
           xlim=xlim,ylim=ylim,useWest=useWest)
borders = importGSHHS(paste0(gshhg,"wdb_borders_i.b"),
           xlim=xlim,ylim=ylim,useWest=useWest,maxLevel=1)
if(exists("land")){
  plotMap(land,xlim=xlim-ifelse(useWest,360,0),ylim=ylim,
    col="lemonchiffon",bg="aliceblue")
  if(!is.null(rivers)) addLines(rivers,col="blue")
  if(!is.null(borders)) addLines(borders,col="red",lwd=2)
}
```

```
## End(Not run)
```

---

importLocs   *Import LocationSet from Text File*

Description

Import a text file and convert it into a LocationSet.

Usage

```
importLocs(LocationSet)
```

Arguments

LocationSet   character – file name of LocationSet text file.

Value

An imported LocationSet.

Author(s)

[Nicholas M. Boers](), Staff Software Engineer
Jobber, Edmonton AB
Last modified Rd: 2023-10-30

See Also

[importPolys](), [importEvents](), [importGSHHS]()

---

importPolys   *Import PolySet from Text File*

Description

Import a text file and convert it into a PolySet with optional PolyData attribute.

Usage

```
importPolys(PolySet, PolyData=NULL, projection=NULL, zone=NULL)
```

Arguments

PolySet	character – file name of PolySet text file.
PolyData	character – optional file name of PolyData text file.
projection	character – optional projection attribute to add to EventData.
zone	numeric – optional zone attribute to add to EventData.

Value

An imported PolySet with optional PolyData attribute.

Author(s)

[Nicholas M. Boers](#), Staff Software Engineer
Jobber, Edmonton AB
Last modified Rd: 2023-10-30

See Also

[importEvents](#), [importLocs](#), [importGSHHS](#)

---

isConvex	*Determine Whether Polygons are Convex*

Description

Determine whether polygons found in a [PolySet](#) are convex.

Usage

```
isConvex (polys)
```

Arguments

polys	[PolySet](#) to use.

Details

Convex polygons do not self-intersect. In a convex polygon, only the first and last vertices may share the same coordinates (i.e., the polygons are optionally closed).

The function does not give special consideration to holes. It returns a value for each unique (PID, SID), regardless of whether a contour represents a hole.

Value

[PolyData](#) with columns PID, SID (*may be missing*), and convex. Column convex contains Boolean values.

Author(s)

[Nicholas M. Boers](#), Staff Software Engineer
Jobber, Edmonton AB
Last modified Rd: 2013-04-10

See Also

[isIntersecting](#), [PolySet](#).

Examples

```
local(envir=.PBSmapEnv,expr={
  #--- load the data (if using R)
  if (!is.null(version$language) && (version$language=="R"))
    data(nepacLL,envir=.PBSmapEnv)
  #--- calculate then print the polygons that are convex
  p <- isConvex(nepacLL);
  #--- nepacLL actually contains no convex polygons
  print(p[p$convex,])
})
```

---

isIntersecting	*Determine Whether Polygons are Self-Intersecting*

Description

Determine whether polygons found in a [PolySet](#) are self-intersecting.

Usage

```
isIntersecting (polys, numericResult = FALSE)
```

Arguments

polys	[PolySet](#) to use.
numericResult	Boolean value; if TRUE, returns the number of intersections.

Details

When numericResult = TRUE, this function counts intersections as the algorithm processes them. It counts certain types (i.e., those involving vertices and those where an edge retraces over an edge) more than once.

The function does not give special consideration to holes. It returns a value for each unique (PID, SID), regardless of whether a contour represents a hole.

Value

[PolyData](#) with columns PID, SID (*may be missing*), and intersecting. If numericResult is TRUE, intersecting contains the number of intersections. Otherwise, it contains a Boolean value.

Author(s)

Nicholas M. Boers, Staff Software Engineer
Jobber, Edmonton AB
Last modified Rd: `2013-04-10`

See Also

isConvex, PolySet.

Examples

```
local(envir=.PBSmapEnv,expr={
  #--- load the data (if using R)
  if (!is.null(version$language) && (version$language=="R"))
  data(nepacLL,envir=.PBSmapEnv)
  #--- calculate then print the polygons that are self-intersecting
  p <- isIntersecting(nepacLL, numericResult = FALSE)
  print(p[p$intersecting,])
})
```

---

joinPolys   *Join One or Two PolySets using a Logic Operation*

Description

Join one or two PolySets using a logic operation.

Usage

```
joinPolys(polysA, polysB=NULL, operation="INT")
```

Arguments

polysA	PolySet to join.
polysB	optional second PolySet with which to join.
operation	one of "DIFF", "INT", "UNION", or "XOR", representing difference, intersection, union, and exclusive-or, respectively.

Details

This function interfaces with the first version of the Clipper Library, specifically version 6.2.1 released 2014-10-31, developed by Angus Johnson. Angus now offers Clipper 2 Library. Prior to 2013-03-23, 'joinPolys' used the General Polygon Clipper library by Alan Murta at the University of Manchester. We keep this historic reference to GPC because 'joinPolys' remains faithful to Murta's definition of a generic polygon, which we describe below.

Murta (2004) defines a *generic polygon* (or *polygon set*) as zero or more disjoint boundaries of arbitrary configuration. He relates a *boundary* to a contour, where each may be convex, concave or

self-intersecting. In a PolySet, the polygons associated with each unique PID loosely correspond to a generic polygon, as they can represent both inner and outer boundaries. Our use of the term *generic polygon* includes the restrictions imposed by a PolySet. For example, the polygons for a given PID cannot be arranged arbitrarily.

If 'polysB' is NULL, this function sequentially applies the logic 'operation' between the generic polygons in 'polysA'. For example, suppose 'polysA' contains three generic polygons (A, B, C). The function outputs the PolySet containing ((A *op* B) *op* C).

If 'polysB' is not NULL, this function applies the logic 'operation' between each generic polygon in 'polysA' and each one in 'polysB'. For example, suppose 'polysA' contains two generic polygons (A, B) and 'polysB' contains two generic polygons (C, D). The function's output is the concatenation of A *op* C, B *op* C, A *op* D, B *op* D, with PIDs 1 to 4, respectively. Generally there are *n* times *m* comparisons, where *n* = number of polygons in 'polysA' and *m* = number of polygons in 'polysB'. If 'polysB' contains only one generic polygon, the function maintains the PIDs from 'polysA'. It also maintains them when 'polysA' contains only one generic polygon and the 'operation' is "DIFF" (difference). Otherwise, if 'polysA' contains only one generic polygon, it maintains the PIDs from 'polysB'.

Value

If 'polysB' is NULL, the resulting PolySet contains a single generic polygon (one PID), possibly with several components (SIDs). The function recalculates the PID and SID columns.

If 'polysB' is not NULL, the resulting PolySet contains one or more generic polygons (PIDs), each with possibly several components (SIDs). The function recalculates the SID column, and depending on the input, it may recalculate the PID column.

Author(s)

C code: Angus Johnson, Computer Programmer

Implementation: Nicholas M. Boers, Staff Software Engineer
Jobber, Edmonton AB

Maintainer: Rowan Haigh, Program Head – Offshore Rockfish
Pacific Biological Station (PBS), Fisheries & Oceans Canada (DFO), Nanaimo BC
locus opus: Remote office, Vancouver BC
Last modified Rd: 2024-09-03

References

Murta, A. (2004) *A General Polygon Clipping Library*. Accessed: Jul 29, 2004.

Johnson, A. (2014) *Clipper* – an open source freeware library for clipping and offsetting lines and polygons. Accessed: Oct 31, 2014.

See Also

In package **PBSmapping**:
addPolys, appendPolys, clipPolys, closePolys, fixBound, fixPOS, locatePolys, plotMap, plotPoints, thickenPolys, thinPolys

Examples

```
local(envir=.PBSmapEnv,expr={
  oldpar = par(no.readonly=TRUE)
  #--- load the data (if using R)
  if (!is.null(version$language) && (version$language=="R"))
    data(nepacLL,envir=.PBSmapEnv)

  ### Example 1. Cut a triangle out of Vancouver Island
  par(mfrow=c(1,1))
  #--- create a triangle to use in clipping
  polysB <- data.frame(PID=rep(1, 3), POS=1:3,
    X=c(-127.5, -124.5, -125.6), Y = c(49.2, 50.3, 48.6))
  #--- intersect nepacLL with the single polygon, and plot the result
  plotMap(joinPolys(nepacLL, polysB), col="cyan")
  #--- add nepacLL in a different line type to emphasize the intersection
  addPolys(nepacLL, border="purple", lty=3, density=0)
  box()

  ### Example 2. Cut Texada and Lasqueti Islands out of Boxes
  xlim = list(box1=c(-124.8,-124),box2=c(-124,-123.9))
  ylim = list(box1=c(49.4,49.85), box2=c(49.85,49.9))
  Xlim = extendrange(xlim); Ylim=extendrange(ylim)
  polyA = as.PolySet(data.frame(
    PID = rep(1:2,each=4), POS = rep(1:4,2),
    X = as.vector(sapply(xlim,function(x){x[c(1,1,2,2)]})),
    Y = as.vector(sapply(ylim,function(x){x[c(1,2,2,1)]}))
    ), projection="LL")
  data(nepacLLhigh,envir=.PBSmapEnv)
  polyB = nepacLLhigh[is.element(nepacLLhigh$PID,c(736,1912)),]
  polyC = joinPolys(polyA, polyB, "DIFF")
  par(mfrow=c(2,2),cex=1,mgp=c(2,0.5,0))
  plotMap(polyA,col="lightblue",xlim=Xlim,ylim=Ylim)
  addPolys(polyB,col="gold");
  text(mean(Xlim)-0.05,Ylim-0.04,"Boxes (A,B)  and  Isles (C,D)")
  labs = calcCentroid(polyA)
  labs[1,c("X","Y")] = labs[2,c("X","Y")]+c(-0.1,-0.05)
  text(labs[,"X"],labs[,"Y"],c("A","B"),font=2)
  plotMap(polyC[is.element(polyC$PID,1),],col="pink",xlim=Xlim,ylim=Ylim)
  text(mean(Xlim)-0.05,Ylim-0.04,"Box A  \"DIFF\"  Isle C")
  plotMap(polyC[is.element(polyC$PID,3),],col="green",xlim=Xlim,ylim=Ylim)
  text(mean(Xlim)-0.05,Ylim-0.04,"Box A  \"DIFF\"  Isle D")
  plotMap(polyC[is.element(polyC$PID,c(1,3)),],col="cyan",xlim=Xlim,ylim=Ylim)
  text(mean(Xlim)-0.05,Ylim-0.04,"Box A  \"DIFF\"  Isles (C,D)")
  par(oldpar)
})
```

---

locateEvents   *Locate Events on the Current Plot*

Description

Locate events on the current plot (using the [locator](#) function).

Usage

```
locateEvents (EID, n = 512, type = "p", ...)
```

Arguments

EID	vector of event IDs (*optional*).
n	maximum number of events to locate.
type	one of "n", "p", "l", or "o". If "p" or "o", then the points are plotted; if "l" or "o", then the points are joined by lines.
...	additional [par](#) parameters for the [locator](#) function.

Details

This function allows its user to define events with mouse clicks on the current plot via the [locator](#) function. The arguments n and type are the usual parameters of the [locator](#) function. If EID is not missing, then n = length(EID).

On exit from [locator](#), suppose the user defined *m* events. If EID was missing, then the output data frame will contain *m* events. However, if EID exists, then the output data frame will contain length(EID) events, and both X and Y will be NA for events EID[(*m*+1):n]. The [na.omit](#) function can remove rows with NAs.

Value

[EventData](#) with columns EID, X, and Y, and projection attribute equal to the map's projection. The function does not set the zone attribute.

Author(s)

[Nicholas M. Boers](#), Staff Software Engineer
Jobber, Edmonton AB
Last modified Rd: 2007-06-06

See Also

[addPoints](#), [combineEvents](#), [convDP](#), [EventData](#), [findCells](#), [findPolys](#), [plotPoints](#).

Examples

```
#--- define five events on the current plot, numbering them 10 to 14
## Not run: events <- locateEvents(EID = 10:14)
```

---

locatePolys   *Locate Polygons on the Current Plot*

Description

Locate polygons on the current plot (using the [locator](#) function).

Usage

```
locatePolys (pdata, n = 512, type = "o", ...)
```

Arguments

pdata   [PolyData](#) (*optional*) with columns PID and SID (*optional*), with two more op-
             tional columns n and type.

n   maximum number of points to locate.

type   one of "n", "p", "l", or "o". If "p" or "o", then the points are plotted; if "l" or
             "o", then the points are joined by lines.

...   additional [par](#) parameters for the [locator](#) function.

Details

This function allows its user to define polygons with mouse clicks on the current plot via the
[locator](#) function. The arguments n and type are the usual parameters for the [locator](#) function,
but the user can specify them for each individual (PID, SID) in a pdata object.

If a pdata object exists, the function ignores columns other than PID, SID, n, and type. If pdata
includes n, then an outer boundary has n > 0 and an inner boundary has n < 0.

On exit from [locator](#), suppose the user defined *m* vertices for a given polygon. For that poly-
gon, the X and Y columns will contain NAs where POS = (*m*+1):n for outer-boundaries and POS =
(|n|-*m*):1 for inner-boundaries. The [na.omit](#) function can remove rows with NAs.

If a pdata object does not exist, the output contains only one polygon with a PID equal to 1. One
inner-boundary polygon (POS goes from n to 1) can be generated by supplying a negative n.

If type = "o" or type = "l", the function draws a line connecting the last and first vertices.

Value

[PolySet](#) with projection attribute equal to the map's projection. The function does not set the
zone attribute.

Author(s)

[Nicholas M. Boers](#), Staff Software Engineer
Jobber, Edmonton AB
Last modified Rd: 2007-06-06

See Also

addPolys, appendPolys, clipPolys, closePolys, findCells, findPolys, fixPOS, joinPolys, plotMap, plotPolys, thickenPolys, thinPolys.

Examples

```
#--- define one polygon with up to 5 vertices on the current plot
## Not run: polys <- locatePolys(n = 5)
```

LocationSet   *LocationSet Objects*

Description

A LocationSet comprises a data frame that summarises which EventData points (EID) lie in which PolySet polygons (PID) or (PID, SID). Events not located in target polygons are not reported. If an event lies on a polygon boundary, an additional LocationSet field called Bdry is set to TRUE. One event can also occur in multiple polygons.

PBSmapping functions that expect LocationSet's will accept properly formatted data frames in their place (see 'Details').

as.LocationSet attempts to coerce a data frame to an object with class LocationSet.

is.LocationSet returns TRUE if its argument is of class LocationSet.

Usage

```
as.LocationSet(x)
is.LocationSet(x, fullValidation = TRUE)
```

Arguments

x   data frame to be coerced or tested.

fullValidation   Boolean value; if TRUE, fully test x.

Details

A PolySet can define regional boundaries for drawing a map, and EventData can give event points on the map. Which events occur in which regions? Our function findPolys resolves this problem. The output lies in a LocationSet, a data frame with three or four columns (EID, PID, SID, Bdry), where SID may be missing. One row in a LocationSet means that the event EID occurs in the polygon (PID, SID). The boundary (Bdry) field specifies whether (Bdry=T) or not (Bdry=F) the event lies on the polygon boundary. If SID refers to an inner polygon boundary, then EID occurs in (PID, SID) only if Bdry=T. An event may occur in multiple polygons. Thus, the same EID can occur in multiple records. If an EID does not fall in any (PID, SID), or if it falls within a hole, it does not occur in the output LocationSet. Inserting the string "LocationSet" as the first element of a LocationSet's class attribute alters the behaviour of some functions, including print (if PBSprint is TRUE) and summary.

Value

The `as.LocationSet` method returns an object with classes `"LocationSet"` and `"data.frame"`, in that order.

Author(s)

[Nicholas M. Boers](#), Staff Software Engineer
Jobber, Edmonton AB
Last modified Rd: 2015-04-23

See Also

[PolySet](#), [PolyData](#), [EventData](#)

---

makeGrid	*Make a Grid of Polygons*

Description

Make a grid of polygons, using `PID`s and `SID`s according to the input arguments.

Usage

```
makeGrid(x, y, byrow=TRUE, addSID=TRUE,
    projection=NULL, zone = NULL, type="rectangle")
```

Arguments

x	numeric – vector of X-coordinates (of length m).
y	numeric – vector of Y-coordinates (of length n).
byrow	logical – if TRUE *and* type='rectangle', increment PID along X (column-wise); – if TRUE *and* type='hexagon', create flat-topped hexagons contiguous by column and increment PID by column; – if FALSE *and* type='hexagon', create pointy-topped hexagons contiguous by row and increment PID by row.
addSID	logical – if TRUE, include an SID field in the resulting [PolySet](#), incremented by the alternative dimension used by PID.
projection	character – optional projection attribute to add to the PolySet.
zone	numeric – optional zone attribute to add to the PolySet.
type	character – type of regular tesselation; choices: "rectangle" or "hexagon".

Details

This function makes a grid of polygons, labeling them according to byrow and addSID.

For rectangular tesselations (grid cells), the variables i and j indicate column and row numbers, respectively, where the lower-left cell of the grid is (1, 1):

- byrow = TRUE and addSID = FALSE implies PID $= i + (j - 1) \times (m - 1)$
- byrow = FALSE and addSID = FALSE implies PID $= j + (i - 1) \times (n - 1)$
- byrow = TRUE and addSID = TRUE implies PID $= i$, SID $= j$
- byrow = FALSE and addSID = TRUE implies PID $= j$, SID $= i$

For hexagonal tesselations (grid cells), i indicates columns for flat-topped hexagons and rows for pointy-topped hexagons. The reverse is true for j. Stemming from their six-sided nature, hexagons will adjoin along a long-edge by row when their orientation is such that one vertex is higher than all the others. Hexagons will adjoin along a long-edge by column when their orientation shows two uppermost vertices.

Value

PolySet with columns PID, SID (*if* addSID=TRUE), POS, X, and Y.
The PolySet is a set of rectangular grid cells when type='rectangle', with vertices:
$(x_i, y_j), (x_{i+1}, y_j), (x_{i+1}, y_{j+1}), (x_i, y_{j+1})$.
The PolySet is a set of hexagonal grid cells when type='hexagon'.

Author(s)

Nicholas M. Boers, Staff Software Engineer
Jobber, Edmonton AB

Rowan Haigh, Program Head – Offshore Rockfish
Pacific Biological Station (PBS), Fisheries & Oceans Canada (DFO), Nanaimo BC
locus opus: Institute of Ocean Sciences (IOS), Sidney BC
Last modified Rd: 2019-01-04

See Also

addPolys, clipPolys, combineEvents, findCells, findPolys, thickenPolys.

Examples

```
local(envir=.PBSmapEnv,expr={
  oldpar = par(no.readonly=TRUE)
  ##--- make a 10 x 10 grid
  polyGrid <- makeGrid(x=0:10, y=0:10)
  ##--- plot the grid
  plotPolys(polyGrid, density=0, projection=1)
  par(oldpar)
})
```

---

makeProps   *Make Polygon Properties*

Description

Append a column for a polygon property (e.g., border or lty) to PolyData based on measurements in the PolyData's Z column.

Usage

```
makeProps(pdata,breaks,propName="col",propVals=1:(length(breaks)-1))
```

Arguments

pdata	PolyData with a Z column.
breaks	either a vector of cut points or a scalar denoting the number of intervals that Z is to be cut into.
propName	name of the new column to append to pdata.
propVals	vector of values to associate with Z breaks.

Details

This function acts like the cut function to produce PolyData suitable for the polyProps plotting argument (see addLabels, addLines, addPoints, addPolys, addStipples, plotLines, plotMap,plotPoints, and plotPolys). The Z column of pdata is equivalent to the data vector x of the cut function.

Value

PolyData with the same columns as pdata plus an additional column propName.

Author(s)

Nicholas M. Boers, Staff Software Engineer
Jobber, Edmonton AB
Last modified Rd: 2013-04-10

See Also

addLabels, addLines, addPoints, addPolys, addStipples, plotLines, plotMap, plotPoints, plotPolys, PolyData, PolySet.

Examples

```
local(envir=.PBSmapEnv,expr={
  #--- create a PolyData object
  pd <- data.frame(PID=1:10, Z=1:10)

  #--- using 3 intervals, create a column named `col' and populate it with
  #--- the supplied values
  makeProps(pdata=pd, breaks=3, propName="col", propVals=c(1:3))
})
```

---

makeTopography   *Make Topography Data from Online Source*

Description

Make topography data suitable for the `'graphics::contour'` and `'grDevices::contourLines'` functions using freely available global seafloor topography data.

Usage

```
makeTopography (dat, digits=2, func=NULL)
```

Arguments

dat	data.frame – data with three optionally-named columns: x, y, and z. The columns must appear in that order.
digits	numeric – integer indicating the precision to be used by the function round on (x,y) values.
func	function – to summarize z if (x,y) points are duplicated. Defaults to mean() if no function is specified.

Details

Suitable data can be obtained through the [Topex acquisition form](). The function `'utils::read.table'` will import dowloaded ASCII files into R, creating objects suitable for the argument `'dat'` in `'makeTopography'`.

When creating data for regions with longitude values spanning -180° to 0°, consider subtracting 360 from the result's longitude coordinates (x).

When creating bathymetry data, consider negating the result's elevations (z) to give depths positive values.

Combinations of (x,y) do not need to be complete (z[x,y]=NA) or unique (z[x,y] = func(z[x,y])).

Value

List with elements x, y, and z. Elements x and y are vectors, while z is a matrix with rownames x and colnames y. The functions `'graphics::contour'` and `'grDevices::contourLines'` expect data conforming to this list format.

Author(s)

Nicholas M. Boers, Staff Software Engineer
Jobber, Edmonton AB

Maintainer: Rowan Haigh, Program Head – Offshore Rockfish
Pacific Biological Station (PBS), Fisheries & Oceans Canada (DFO), Nanaimo BC
locus opus: Institute of Ocean Sciences (IOS), Sidney BC
Last modified Rd: 2021-01-11

See Also

In package **graphics**:
contour
In package **grDevices**:
contourLines
In package **PBSmapping**:
convCP

Examples

```
local(envir=.PBSmapEnv,expr={
  oldpar = par(no.readonly=TRUE)
  #--- Example 1: Sample data frame and conversion.
  file <- data.frame(X=c(1,1,2,2),Y=c(3,4,3,4),Z=c(5,6,7,8))
  print(makeTopography(file))

  #--- Example 2: Aleutian Islands bathymetry
  isob <- c(100,500,1000,2500,5000)
  icol <- rgb(0,0,seq(255,100,len=length(isob)),max=255)
  afile <- paste(system.file(package="PBSmapping"),
    "/Extra/aleutian.txt",sep="")
  aleutian <- read.table(afile, header=FALSE, col.names=c("x","y","z"))
  aleutian$x <- aleutian$x - 360
  aleutian$z <- -aleutian$z
  alBathy <- makeTopography(aleutian)
  alCL <- contourLines(alBathy,levels=isob)
  alCP <- convCP(alCL)
  alPoly <- alCP$PolySet
  attr(alPoly,"projection") <- "LL"
  plotMap(alPoly, type="n", cex.axis=1.2, cex.lab=1.5)
  addLines(alPoly,col=icol)
  data(nepacLL,envir=.PBSmapEnv)
  addPolys(nepacLL,col="gold")
  legend(x="topleft",bty="n",col=icol,lwd=2,legend=as.character(isob))
  par(oldpar)
})
```

---

nepacLL   *Data: Shorelines of the NE Pacific Ocean and of the World*

Description

PolySet of polygons for the shorelines of the northeast Pacific Ocean and of the world, both in normal and high resolution.

Usage

```
data(nepacLL)
data(nepacLLhigh)
data(worldLL)
data(worldLLhigh)
```

Format

Data frame consisting of 4 columns: PID = primary polygon ID, POS = position of each vertex within a given polygon, X = longitude coordinate, and Y = latitude coordinate. Attributes: projection = "LL".

Note

In R, the data must be loaded using the data function.

Author(s)

Nicholas M. Boers, Staff Software Engineer
Jobber, Edmonton AB
Last modified Rd: 2023-10-30

Source

Polygon data from the GSHHG (Global Self-consistent, Hierarchical, High-resolution Geography) Database.
Download the native binary files of shoreline polygons, rivers, and borders contained in the latest zip archive (version 2.3.4) at
(www.soest.hawaii.edu/pwessel/gshhg).

```
nepacLL <- importGSHHS("gshhs_h.b", xlim=c(-190,-110), ylim=c(34,72),
         level=1, n=15, xoff=-360)

nepacLLhigh <- importGSHHS("gshhs_f.b", xlim=c(-190,-110),
              ylim=c(34,72), level=1, n=0, xoff=-360)
nepacLLhigh <- thinPolys(nepacLLhigh, tol=0.1, filter=3)

worldLL <- importGSHHS("gshhs_l.b", xlim=c(-20,360), ylim=c(-90,90),
```

```
            level=1, n=15, xoff=0)
worldLL <- PBSmapping:::.fixGSHHSWorld(worldLL)

worldLLhigh <- importGSHHS("gshhs_i.b", xlim=c(-20,360),
               ylim=c(-90,90), level=1, n=15, xoff=0)
worldLLhigh <- PBSmapping:::.fixGSHHSWorld(worldLLhigh)
```

References

Wessel, P. and Smith, W.H.F. (1996) A global, self-consistent, hierarchical, high-resolution shore-line database. *Journal of Geophysical Research* **101**, 8741–8743.
(www.soest.hawaii.edu/pwessel/gshhg/Wessel+Smith_1996_JGR.pdf)

See Also

Data:
[bcBathymetry](), [surveyData](), [towData]()

Functions:
[importGSHHS](), [plotMap](), [plotPolys](), [addPolys](), [clipPolys](), [refocusWorld](), [thickenPolys](), [thinPolys]()

PBSmapping   *PBS Mapping: Draw Maps and Implement Other GIS Procedures*

Description

This software has evolved from fisheries research conducted at the Pacific Biological Station (PBS) in Nanaimo, British Columbia, Canada. It extends the R language to include two-dimensional plotting features similar to those commonly available in a Geographic Information System (GIS). Embedded C code speeds algorithms from computational geometry, such as finding polygons that contain specified point events or converting between longitude-latitude and Universal Transverse Mercator (UTM) coordinates. It includes data for a global shoreline and other data sets in the public domain.

For a complete user's guide, see the file `PBSmapping-UG.pdf` in the R directory `.../library/PBSmapping/doc`.

`PBSmapping` includes 10 demos that appear as figures in the User's Guide. To see them, run the function `.PBSfigs()`.

More generally, a user can view all demos available from locally installed packages with the function `runDemos()` in our related (and recommended) package `PBSmodelling`.

Author(s)

[Nicholas M. Boers](), Staff Software Engineer
Jobber, Edmonton AB
Last modified `Rd: 2008-09-03`

PBSprint   *Specify Whether to Print Summaries*

Description

Specify whether PBS Mapping should print object summaries or not. If not, data objects are displayed as normal.

Usage

```
PBSprint
```

Details

If `PBSprint = TRUE`, the mapping software will print summaries rather than the data frames for EventData, LocationSet, PolyData, and PolySet objects. If `PBSprint = FALSE`, it will print the data frames.

This variable's default value is `FALSE`.

Value

`TRUE` or `FALSE`, depending on the user's preference.

Author(s)

[Nicholas M. Boers](#), Staff Software Engineer
Jobber, Edmonton AB
Last modified Rd: `2007-06-06`

See Also

[summary](#).

---

placeHoles   *Place Holes Under Solids*

Description

Place secondary polygons identified as holes (inner contours) under primary polygons identified as solids (outer contours) if the vertices of the holes lie completely within the vertices of the solids. This operation is performed for each primary polygon until all holes have been assigned.

Usage

```
placeHoles(polyset, minVerts=3,
   orient=FALSE, show.progress=FALSE, unique.pid=FALSE)
```

Arguments

polyset	a valid **PBSmapping** PolySet.
minVerts	numeric – minimum number of vertices required for a polygon representing a hole to be retained (does not affect solids).
orient	logical – if TRUE, use each polygon's rotation to determine its nature: clockwise = solid (outer contour), counter-clockwise = hole (inner contour).
show.progress	logical – if TRUE, display on the command console the progress of placing holes under solids.
unique.pid	logical – if TRUE, ignore the input PIDs and redefine them from 1 to the number of solids; this inherently destroys the previous organisation that a creator may have intended for the PolySet.

Details

The algorithm identifies outer contours (solids) and inner contours (holes) using either the default PBSmapping method (solids = increasing ″POS″, holes = decreasing ″POS″) or the rotational direction of the polygons (solids = clockwise, holes = counter-clockwise).

It then systematically starts matching holes with solids based on their vertices being completely within the boundaries of the solid. If a hole happens to match a current solid completley (all vertices on the boundary), then the hole is not matched to this solid because it is a hole in another solid that creates space for the current solid.

To facilitate computation, the algorithm assumes that once a hole is located in a solid, it will not occur in any other solid. This means that for each successive solid, the number of candidate holes will either decrease or stay the same.

This function makes use of the PBSmapping hidden function ″.is.in″ which uses the C code ″findPolys″. The latter only returns events found in a polygon (or on the boundary) but .is.in evaluates all events and returns a list containing:
″e.in″ – events within the polygon,
″e.out″ – events outside the polygon,
″all.in″ – logical value of whether all events are in the polygon,
″all.out″ – logical value of whether all events are outside the polygon,
″all.bdry″ – logical value of whether all events occur on the boundary of the polygon

Value

Returns the input PolySet where holes have been arranged beneath appropriate solids for each PID (original or redefined).

Author(s)

Rowan Haigh, Program Head – Offshore Rockfish
Pacific Biological Station (PBS), Fisheries & Oceans Canada (DFO), Nanaimo BC
locus opus: Offsite, Vancouver BC
Last modified Rd: 2023-10-30

See Also

In package **PBSmapping**:
[findPolys](), [is.PolySet](), [dot-is.in]()

---

plotLines   *Plot a PolySet as Polylines*

Description

Plot a [PolySet]() as polylines.

Usage

```
plotLines (polys, xlim = NULL, ylim = NULL, projection = FALSE,
           plt = c(0.11, 0.98, 0.12, 0.88), polyProps = NULL,
           lty = NULL, col = NULL, bg = 0, axes = TRUE,
           tckLab = TRUE, tck = 0.014, tckMinor = 0.5 * tck, ...)
```

Arguments

polys	[PolySet]() to plot (*required*).
xlim	range of X-coordinates.
ylim	range of Y-coordinates.
projection	desired projection when [PolySet]() lacks a projection attribute; one of "LL", "UTM", or a numeric value. If Boolean, specifies whether to check polys for a projection attribute.
plt	four element numeric vector (x1, x2, y1, y2) giving the coordinates of the plot region measured as a fraction of the figure region. Set to NULL if mai in par is desired.
polyProps	[PolyData]() specifying which polylines to plot and their properties. [par]() parameters passed as direct arguments supersede these data.
lty	vector describing line types (cycled by PID).
col	vector describing colours (cycled by PID).
bg	background colour of the plot.
axes	Boolean value; if TRUE, plot axes.
tckLab	Boolean vector (length 1 or 2); if TRUE, label the major tick marks. If given a two-element vector, the first element describes the tick marks on the x-axis and the second element describes those on the y-axis.
tck	numeric vector (length 1 or 2) describing the length of tick marks as a fraction of the smallest dimension. If tckLab = TRUE, these tick marks will be automatically labelled. If given a two-element vector, the first element describes the tick marks on the x-axis and the second element describes those on the y-axis.

tckMinor	numeric vector (length 1 or 2) describing the length of tick marks as a fraction of the smallest dimension. These tick marks can not be automatically labelled. If given a two-element vector, the first element describes the tick marks on the x-axis and the second element describes those on the y-axis.
...	additional [par](#) parameters, or the arguments main, sub, xlab, or ylab for the [title](#) function.

Details

This function plots a [PolySet](#), where each unique (PID, SID) describes a polyline. It does not connect each polyline's last vertex to its first. Unlike [plotMap](#), the function ignores the aspect ratio. It clips polys to xlim and ylim before plotting.

The function creates a blank plot when polys equals NULL. In this case, the user must supply both xlim and ylim arguments. Alternatively, it accepts the argument type = ″n″ as part of ..., which is equivalent to specifying polys = NULL, but requires a [PolySet](#). In both cases, the function's behaviour changes slightly. To resemble the [plot](#) function, it plots the border, labels, and other parts according to [par](#) parameters such as col.

For additional help on the arguments lty and col, please see [par](#).

Value

[PolyData](#) consisting of the PolyProps used to create the plot.

Note

To satisfy the aspect ratio, this plotting routine resizes the plot region. Consequently, [par](#) parameters such as plt, mai, and mar will change. When the function terminates, these changes persist to allow for additions to the plot.

Author(s)

[Nicholas M. Boers](#), Staff Software Engineer
Jobber, Edmonton AB
Last modified Rd: 2013-04-10

See Also

[addLines](#), [calcLength](#), [clipLines](#), [closePolys](#), [convLP](#), [fixBound](#), [fixPOS](#),
[locatePolys](#), [thinPolys](#), [thickenPolys](#).

Examples

```
local(envir=.PBSmapEnv,expr={
  oldpar = par(no.readonly=TRUE)
  #--- create a PolySet to plot
  polys <- data.frame(PID=rep(1,4),POS=1:4,X=c(0,1,1,0),Y=c(0,0,1,1))
  #--- plot the PolySet
  plotLines(polys, xlim=c(-.5,1.5), ylim=c(-.5,1.5))
  par(oldpar)
})
```

---

plotMap   *Plot a PolySet as a Map*

Description

Plot a [PolySet](#) as a map, using the correct aspect ratio.

Usage

```
plotMap(polys, xlim = NULL, ylim = NULL, projection = TRUE,
        plt = c(0.11, 0.98, 0.12, 0.88), polyProps = NULL,
        border = NULL, lty = NULL, col = NULL, colHoles = NULL,
        density = NA, angle = NULL, bg = 0, axes = TRUE,
        tckLab = TRUE, tck = 0.014, tckMinor = 0.5 * tck, ...)

.plotMaps(polys, xlim, ylim, projection, plt, polyProps,
          border, lty, col, colHoles, density, angle, bg,
          axes, tckLab, tck, tckMinor, isType, ...)

.initPlotRegion(projection, xlim, ylim, plt)
```

Arguments

polys	data.frame – [PolySet](#) to plot (*required*).
xlim	numeric – range of X-coordinates.
ylim	numeric – range of Y-coordinates.
projection	character|logical – desired projection when [PolySet](#) lacks a projection attribute; one of "LL", "UTM", or a numeric value. If Boolean, specifies whether to check polys for a projection attribute.
plt	numeric – four element vector (x1, x2, y1, y2) giving the coordinates of the plot region measured as a fraction of the figure region. Set to NULL if mai in par is desired.
polyProps	data.frame – [PolyData](#) specifying which polygons to plot and their properties. [par](#) parameters passed as direct arguments supersede these data.
border	numeric|character – vector describing edge colours (cycled by PID).
lty	numeric|character – vector describing line types (cycled by PID).
col	numeric|character – vector describing fill colours (cycled by PID).
colHoles	numeric|character – vector describing hole colours (cycled by PID). The default, NULL, should be used in most cases as it renders holes transparent. colHoles is designed solely to eliminate retrace lines when images are converted to PDF format. If colHoles is specified, underlying information (i.e., previously plotted shapes) will be obliterated. If NA is specified, only outer polygons are drawn, consequently filling holes.

density	numeric – vector describing shading line densities (lines per inch, cycled by PID).
angle	numeric – vector describing shading line angles (degrees, cycled by PID).
bg	numeric|character – background colour of the plot
axes	logical – if TRUE, plot axes
tckLab	logical – vector (length 1 or 2); if TRUE, label the major tick marks. If given a two-element vector, the first element describes the tick marks on the x-axis and the second element describes those on the y-axis.
tck	numeric – vector (length 1 or 2) describing the length of tick marks as a fraction of the smallest dimension. If tckLab = TRUE, these tick marks will be automatically labelled. If given a two-element vector, the first element describes the tick marks on the x-axis and the second element describes those on the y-axis.
tckMinor	numeric – vector (length 1 or 2) describing the length of tick marks as a fraction of the smallest dimension. These tick marks can not be automatically labelled. If given a two-element vector, the first element describes the tick marks on the x-axis and the second element describes those on the y-axis.
...	dots – additional [par](par) parameters, or the arguments main, sub, xlab, or ylab for the title function.
isType	character – one of 'points', 'polygons', or 'lines'

Details

This function plots a PolySet, where each unique (PID, SID) describes a polygon. It connects each polygon's last vertex to its first. The function supports both borders (border, lty) and fills (col, density, angle). When supplied with the appropriate arguments, it can draw only borders or only fills . Unlike plotLines and plotPolys, it uses the aspect ratio supplied in the projection attribute of polys. If this attribute is missing, it attempts to use its projection argument. In the absence of both, it uses a default aspect ratio of 1:1. It clips polys to xlim and ylim before plotting.

The function creates a blank plot when polys equals NULL. In this case, the user must supply both xlim and ylim arguments. Alternatively, it accepts the argument type = "n" as part of ..., which is equivalent to specifying polys = NULL, but requires a PolySet. In both cases, the function's behaviour changes slightly. To resemble the plot function, it plots the border, labels, and other parts according to [par](par) parameters such as col.

For additional help on the arguments border, lty, col, density, and angle, please see polygon and [par](par).

Value

PolyData consisting of the PolyProps used to create the plot.

Note

To satisfy the aspect ratio, this plotting routine resizes the plot region. Consequently, [par](par) parameters such as plt, mai, and mar will change. When the function terminates, these changes persist to allow for additions to the plot.

Auxiliary dot function '.initPlotRegion' initialises the plot region, accounting for the aspect ratio.

Author(s)

Nicholas M. Boers, Software Engineer, Jobber, Edmonton AB

Maintainer: Rowan Haigh, Program Head – Offshore Rockfish
Pacific Biological Station (PBS), Fisheries & Oceans Canada (DFO), Nanaimo BC
locus opus: Remote office, Vancouver BC
Last modified Rd: 2024-09-03

See Also

In package **PBSmapping**:
addLabels, addPolys, addStipples, clipPolys, closePolys, fixBound, fixPOS, locatePolys, plotLines, plotPoints, thinPolys, thickenPolys.

Examples

```
local(envir=.PBSmapEnv,expr={
  oldpar = par(no.readonly=TRUE)
  #--- create a PolySet to plot
  polys <- data.frame(PID=rep(1,4),POS=1:4,X=c(0,1,1,0),Y=c(0,0,1,1))
  #--- plot the PolySet
  plotMap(polys,xlim=c(-.5,1.5),ylim=c(-.5,1.5),density=0,projection=1)
  par(oldpar)
})
```

---

plotPoints	*Plot EventData/PolyData as Points*

Description

Plot EventData/PolyData, where each unique EID or (PID, SID) describes a point.

Usage

```
plotPoints (data, xlim = NULL, ylim = NULL, projection = FALSE,
            plt = c(0.11, 0.98, 0.12, 0.88), polyProps = NULL,
            cex = NULL, col = NULL, pch = NULL, axes = TRUE,
            tckLab = TRUE, tck = 0.014, tckMinor = 0.5 * tck, ...)
```

Arguments

data	EventData or PolyData to plot (*required*).
xlim	range of X-coordinates.
ylim	range of Y-coordinates.
projection	desired projection when PolySet lacks a projection attribute; one of "LL", "UTM", or a numeric value. If Boolean, specifies whether to check polys for a projection attribute.

plt	four element numeric vector (x1, x2, y1, y2) giving the coordinates of the plot region measured as a fraction of the figure region. Set to NULL if mai in par is desired.
polyProps	[PolyData](#) specifying which points to plot and their properties. [par](#) parameters passed as direct arguments supersede these data.
cex	vector describing character expansion factors (cycled by EID or PID).
col	vector describing colours (cycled by EID or PID).
pch	vector describing plotting characters (cycled by EID or PID).
axes	Boolean value; if TRUE, plot axes.
tckLab	Boolean vector (length 1 or 2); if TRUE, label the major tick marks. If given a two-element vector, the first element describes the tick marks on the x-axis and the second element describes those on the y-axis.
tck	numeric vector (length 1 or 2) describing the length of tick marks as a fraction of the smallest dimension. If tckLab = TRUE, these tick marks will be automatically labelled. If given a two-element vector, the first element describes the tick marks on the x-axis and the second element describes those on the y-axis.
tckMinor	numeric vector (length 1 or 2) describing the length of tick marks as a fraction of the smallest dimension. These tick marks can not be automatically labelled. If given a two-element vector, the first element describes the tick marks on the x-axis and the second element describes those on the y-axis.
...	additional [par](#) parameters, or the arguments main, sub, xlab, or ylab for the [title](#) function.

Details

This function clips data to xlim and ylim before plotting. It only adds [PolyData](#) containing X and Y columns.

The function creates a blank plot when polys equals NULL. In this case, the user must supply both xlim and ylim arguments. Alternatively, it accepts the argument type = ″n″ as part of . . . , which is equivalent to specifying polys = NULL, but requires a [PolySet](#). In both cases, the function's behaviour changes slightly. To resemble the [plot](#) function, it plots the border, labels, and other parts according to [par](#) parameters such as col.

For additional help on the arguments cex, col, and pch, please see [par](#).

Value

[PolyData](#) consisting of the PolyProps used to create the plot.

Note

To satisfy the aspect ratio, this plotting routine resizes the plot region. Consequently, [par](#) parameters such as plt, mai, and mar will change. When the function terminates, these changes persist to allow for additions to the plot.

Author(s)

[Nicholas M. Boers](#), Staff Software Engineer
Jobber, Edmonton AB
Last modified Rd: `2013-04-10`

See Also

[addPoints](#), [combineEvents](#), [convDP](#), [findPolys](#), [locateEvents](#).

Examples

```
local(envir=.PBSmapEnv,expr={
  oldpar = par(no.readonly=TRUE)
  #--- load the data (if using R)
  if (!is.null(version$language) && (version$language=="R"))
    data(nepacLL,surveyData,envir=.PBSmapEnv)
  #--- plot a map
  plotMap(nepacLL, xlim=c(-136, -125), ylim=c(48, 57))
  #--- add events
  addPoints(surveyData, col=1:7)
  par(oldpar)
})
```

---

plotPolys   *Plot a PolySet as Polygons*

Description

Plot a [PolySet](#) as polygons.

Usage

```
plotPolys (polys, xlim = NULL, ylim = NULL, projection = FALSE,
           plt = c(0.11, 0.98, 0.12, 0.88), polyProps = NULL,
           border = NULL, lty = NULL, col = NULL, colHoles = NULL,
           density = NA, angle = NULL, bg = 0, axes = TRUE,
           tckLab = TRUE, tck = 0.014, tckMinor = 0.5 * tck, ...)
```

Arguments

polys	[PolySet](#) to plot (*required*).
xlim	range of X-coordinates.
ylim	range of Y-coordinates.
projection	desired projection when [PolySet](#) lacks a projection attribute; one of `"LL"`, `"UTM"`, or a numeric value. If Boolean, specifies whether to check polys for a projection attribute.

plt	four element numeric vector (`x1`, `x2`, `y1`, `y2`) giving the coordinates of the plot region measured as a fraction of the figure region. Set to `NULL` if `mai` in `par` is desired.
polyProps	[PolyData](#) specifying which polygons to plot and their properties. [par](#) parameters passed as direct arguments supersede these data.
border	vector describing edge colours (cycled by PID).
lty	vector describing line types (cycled by PID).
col	vector describing fill colours (cycled by PID).
colHoles	vector describing hole colours (cycled by PID). The default, `NULL`, should be used in most cases as it renders holes transparent. `colHoles` is designed solely to eliminate retrace lines when images are converted to PDF format. If `colHoles` is specified, underlying information (i.e., previously plotted shapes) will be obliterated. If `NA` is specified, only outer polygons are drawn, consequently filling holes.
density	vector describing shading line densities (lines per inch, cycled by PID).
angle	vector describing shading line angles (degrees, cycled by PID).
bg	background colour of the plot.
axes	Boolean value; if `TRUE`, plot axes.
tckLab	Boolean vector (length 1 or 2); if `TRUE`, label the major tick marks. If given a two-element vector, the first element describes the tick marks on the x-axis and the second element describes those on the y-axis.
tck	numeric vector (length 1 or 2) describing the length of tick marks as a fraction of the smallest dimension. If `tckLab = TRUE`, these tick marks will be automatically labelled. If given a two-element vector, the first element describes the tick marks on the x-axis and the second element describes those on the y-axis.
tckMinor	numeric vector (length 1 or 2) describing the length of tick marks as a fraction of the smallest dimension. These tick marks can not be automatically labelled. If given a two-element vector, the first element describes the tick marks on the x-axis and the second element describes those on the y-axis.
...	additional [par](#) parameters, or the arguments `main`, `sub`, `xlab`, or `ylab` for the [title](#) function.

Details

This function plots a [PolySet](#), where each unique (PID, SID) describes a polygon. It connects each polygon's last vertex to its first. The function supports both borders (`border`, `lty`) and fills (`col`, `density`, `angle`). When supplied with the appropriate arguments, it can draw only borders or only fills. Unlike [plotMap](#), it ignores the aspect ratio. It clips `polys` to `xlim` and `ylim` before plotting.

This function creates a blank plot when `polys` equals `NULL`. In this case, the user must supply both `xlim` and `ylim` arguments. Alternatively, it accepts the argument `type = "n"` as part of ..., which is equivalent to specifying `polys = NULL`, but requires a [PolySet](#). In both cases, the function's behaviour changes slightly. To resemble the [plot](#) function, it plots the border, labels, and other parts according to [par](#) parameters such as `col`.

For additional help on the arguments `border`, `lty`, `col`, `density`, and `angle`, please see [polygon](#) and [par](#).

Value

[PolyData](#) consisting of the `PolyProps` used to create the plot.

Note

To satisfy the aspect ratio, this plotting routine resizes the plot region. Consequently, [par](#) parameters such as `plt`, `mai`, and `mar` will change. When the function terminates, these changes persist to allow for additions to the plot.

Author(s)

[Nicholas M. Boers](#), Staff Software Engineer
Jobber, Edmonton AB
Last modified Rd: `2013-04-10`

See Also

[addLabels](#), [addPolys](#), [addStipples](#), [clipPolys](#), [closePolys](#), [fixBound](#), [fixPOS](#), [locatePolys](#), [plotLines](#), [plotMap](#), [plotPoints](#), [thinPolys](#), [thickenPolys](#).

Examples

```
local(envir=.PBSmapEnv,expr={
  oldpar = par(no.readonly=TRUE)
  #--- create a PolySet to plot
  polys <- data.frame(PID=rep(1,4),POS=1:4,X=c(0,1,1,0),Y=c(0,0,1,1))
  #--- plot the PolySet
  plotPolys(polys, xlim=c(-.5,1.5), ylim=c(-.5,1.5), density=0)
  par(oldpar)
})
```

PolyData   *PolyData Objects*

Description

A PolyData object comprises a data frame that summarises information for each polyline/polygon in a PolySet; each PolyData record is defined by a unique PID or (PID, SID combination).

PBSmapping functions that expect PolyData will accept properly formatted data frames in their place (see 'Details').

`as.PolyData` attempts to coerce a data frame to an object with class PolyData.

`is.PolyData` returns `TRUE` if its argument is of class PolyData.

Usage

```
as.PolyData(x, projection = NULL, zone = NULL)
is.PolyData(x, fullValidation = TRUE)
```

Arguments

x	data frame to be coerced or tested.
projection	optional `projection` attribute to add to PolyData, possibly overwriting an existing attribute.
zone	optional `zone` attribute to add to PolyData, possibly overwriting an existing attribute.
fullValidation	Boolean value; if `TRUE`, fully test `x`.

Details

We define PolyData as a data frame with a first column named `PID` and (optionally) a second column named `SID`. Unlike a PolySet, where each contour has many records corresponding to the vertices, a PolyData object must have only one record for each `PID` or each (`PID`, `SID`) combination. Conceptually, this object associates data with contours, where the data correspond to additional fields in the data frame. The R language conveniently allows data frames to contain fields of various atomic modes (`"logical"`, `"numeric"`, `"complex"`, `"character"`, and `"null"`). For example, PolyData with the fields (PID, PName) might assign character names to a set of primary polygons. Additionally, if fields X and Y exist (perhaps representing locations for placing labels), consider adding attributes `zone` and `projection`. Inserting the string `"PolyData"` as the class attribute's first element alters the behaviour of some functions, including print (if PBSprint is TRUE) and summary.

Our software particularly uses PolyData to set various plotting characteristics. Consistent with graphical parameters used by the R/S functions lines and polygon, column names can specify graphical properties:

- `lty` - line type in drawing the border and/or shading lines;
- `col` - line or fill colour;
- `border` - border colour;
- `density` - density of shading lines;
- `angle` - angle of shading lines.

When drawing polylines (as opposed to closed polygons), only `lty` and `col` have meaning.

Value

The `as.PolyData` method returns an object with classes `"PolyData"` and `"data.frame"`, in that order.

Author(s)

Nicholas M. Boers, Staff Software Engineer
Jobber, Edmonton AB
Last modified Rd: `2015-04-23`

See Also

PolySet, EventData, LocationSet

PolySet   *PolySet Objects*

Description

A PolySet object comprises a data frame that defines a collection of polygonal contours (i.e., line segments joined at vertices). These contours can be open-ended (polylines) or closed (polygons).

PBSmapping functions that expect PolySet's will accept properly formatted data frames in their place (see 'Details').

`as.PolySet` attempts to coerce a data frame to an object with class PolySet.

`is.PolySet` returns TRUE if its argument is of class PolySet.

Usage

```
as.PolySet(x, projection = NULL, zone = NULL)
is.PolySet(x, fullValidation = TRUE)
```

Arguments

x	data frame to be coerced or tested.
projection	optional `projection` attribute to add to the PolySet, possibly overwriting an existing attribute.
zone	optional `zone` attribute to add to the PolySet, possibly overwriting an existing attribute.
fullValidation	Boolean value; if TRUE, fully test x.

Details

In our software, a PolySet data frame defines a collection of polygonal contours (i.e., line segments joined at vertices), based on four or five numerical fields:

- `PID` - the primary identification number for a contour;
- `SID` - optional, the secondary identification number for a contour;
- `POS` - the position number associated with a vertex;
- `X` - the horizontal coordinate at a vertex;
- `Y` - the vertical coordinate at a vertex.

The simplest PolySet lacks an `SID` column, and each `PID` corresponds to a different contour. By analogy with a child's "follow the dots" game, the `POS` field enumerates the vertices to be connected by straight lines. Coordinates (`X`, `Y`) specify the location of each vertex. Thus, in familiar mathematical notation, a contour consists of n points (x_i, y_i) with $i = 1, ..., n$, where i corresponds to the `POS` index. A PolySet has two potential interpretations. The first associates a line segment with each successive pair of points from 1 to n, giving a *polyline* (in GIS terminology) composed of the sequential segments. The second includes a final line segment joining points n and 1, thus giving a *polygon*.

The secondary ID field allows us to define regions as composites of polygons. From this point of view, each primary ID identifies a collection of polygons distinguished by secondary IDs. For example, a single management area (`PID`) might consist of two fishing areas, each defined by a unique `SID`. A secondary polygon can also correspond to an inner boundary, like the hole in a doughnut. We adopt the convention that `POS` goes from 1 to n along an outer boundary, but from n to 1 along an inner boundary, regardless of rotational direction. This contrasts with other GIS software, such as ArcView (ESRI 1996), in which outer and inner boundaries correspond to clockwise and counter-clockwise directions, respectively.

The SID field in a PolySet with secondary IDs must have integer values that appear in ascending order for a given `PID`. Furthermore, inner boundaries must follow the outer boundary that encloses them. The `POS` field for each contour (`PID`, `SID`) must similarly appear as integers in strictly increasing or decreasing order, for outer and inner boundaries respectively. If the `POS` field erroneously contains floating-point numbers, `fixPOS` can renumber them as sequential integers, thus simplifying the insertion of a new point, such as point 3.5 between points 3 and 4.

A PolySet can have a `projection` attribute, which may be missing, that specifies a map projection. In the current version of PBS Mapping, projection can have character values `"LL"` or `"UTM"`, referring to "Longitude-Latitude" and "Universal Transverse Mercator". We explain these projections more completely below. If projection is numeric, it specifies the aspect ratio r, the number of x units per y unit. Thus, r units of x on the graph occupy the same distance as one unit of y. Another optional attribute `zone` specifies the UTM zone (if `projection="UTM"`) or the preferred zone for conversion from Longitude-Latitude (if `projection="LL"`).

A data frame's class attribute by default contains the string `"data.frame"`. Inserting the string `"PolySet"` as the class vector's first element alters the behaviour of some functions. For example, the `summary` function will print details specific to a PolySet. Also, when `PBSprint` is TRUE, the print function will display a PolySet's summary rather than the contents of the data frame.

Value

The `as.PolySet` method returns an object with classes `"PolySet"` and `"data.frame"`, in that order.

Author(s)

Nicholas M. Boers, Staff Software Engineer
Jobber, Edmonton AB
Last modified `Rd: 2022-09-06`

References

Environmental Systems Research Institute (ESRI). (1996) *ArcView GIS: The Geographic Information System for Everyone*. ESRI Press, Redlands, California. 340 pp.

See Also

PolyData, EventData, LocationSet

---

print   *Print PBS Mapping Objects*

Description

This function displays information about a PBS Mapping object.

Functions `'summary.EventData'`, `'summary.LocationSet'`, `'summary.PolyData'`, and `'summary.PolySet'` produce an object with class `'summary.PBS'`.

Usage

```
## S3 method for class 'EventData'
print(x, ...)
## S3 method for class 'LocationSet'
print(x, ...)
## S3 method for class 'PolyData'
print(x, ...)
## S3 method for class 'PolySet'
print(x, ...)
## S3 method for class 'summary.PBS'
print(x, ...)
```

Arguments

x	object – a PBS Mapping object of appropriate `class`.
...	dots – additional arguments to `print`.

Author(s)

[Nicholas M. Boers](#), Staff Software Engineer
Jobber, Edmonton AB
Last modified `Rd: 2019-03-14`

See Also

In package **PBSmapping**:
Data structures: [EventData](#), [LocationSet](#), [PolyData](#), [PolySet](#)
Functions: [PBSprint](#), [summary](#)

Examples

```
local(envir=.PBSmapEnv,expr={
  #--- load the data (if using R)
  if (!is.null(version$language) && (version$language=="R"))
    data(nepacLL,envir=.PBSmapEnv)
  #--- change to summary printing style
  PBSprint <- TRUE
```

```
  #--- print the PolySet
  print(nepacLL)
})
```

---

pythagoras   *Data: Pythagoras' Theorem Diagram PolySet*

Description

[PolySet](#) of shapes to prove Pythagoras' Theorem: $a^2 + b^2 = c^2$.

Usage

```
data(pythagoras)
```

Format

4 column data frame: `PID` = primary polygon ID, `POS` = position of each vertex within a given polyline, `X` = X-coordinate, and `Y` = Y-coordinate. Attributes: `projection = 1`.

Note

In R, the data must be loaded using the [data](#) function.

Author(s)

[Nicholas M. Boers](#), Staff Software Engineer
Jobber, Edmonton AB
Last modified Rd: 2022-09-06

Source

An artificial construct to illustrate the proof of Pythagoras' Theorem using trigonometry.

See Also

[addPolys](#), [plotPolys](#), [plotMap](#), [PolySet](#).

---

refocusWorld   *Refocus the* worldLL/worldLLhigh *Data Sets*

Description

Refocus the worldLL/worldLLhigh data sets, e.g., refocus them so that Eastern Canada appears to the west of Western Europe.

Usage

```
refocusWorld (polys, xlim=NULL, ylim=NULL, clip.AN=TRUE)
```

Arguments

polys	PolySet – object with one or more polygons; typically worldLL or worldLLhigh (*required*).
xlim	numeric – range of X-coordinates.
ylim	numeric – range of Y-coordinates.
clip.AN	logical – if TRUE, clip expanded Antarctica to 'xlim' of refocused polygons other than Antarctica and to user-defined/default 'ylim'.

Details

This function accepts a [PolySet](#) containing one or more polygons with X-coordinates that collectively span approximately 360 degrees. The function effectively joins the [PolySet](#) into a cylinder and then splits it at an arbitrary longitude according to the user-specified limits. Modifications in the resulting [PolySet](#) are restricted to shifting X-coordinates by +/- multiples of 360 degrees, and instead of clipping polygons, the return value simply omits out-of-range polygons.

Value

[PolySet](#), likely a subset of the input [PolySet](#), which retains the same PID/SID values.

Note

The Antarctic polygon is treated as a special case in that it is expanded longitudinally by duplicating it to the West and East of the base polygon. The expanded Antarctica is then clipped to the limits of the plot, or not if 'clip.NA=FALSE'.

Author(s)

[Nicholas M. Boers](#), Staff Software Engineer
Jobber, Edmonton AB

[Rowan Haigh](#), Program Head – Offshore Rockfish
Pacific Biological Station (PBS), Fisheries & Oceans Canada (DFO), Nanaimo BC
locus opus: Institute of Ocean Sciences (IOS), Sidney BC
Last modified Rd: 2018-10-26

See Also

joinPolys

Examples

```
local(envir=.PBSmapEnv,expr={
  oldpar = par(no.readonly=TRUE)
  #--- load appropriate data
  data(worldLL,envir=.PBSmapEnv)
  #--- set limits
  xlim <- c(-100,25)
  ylim <- c(0,90)
  #--- refocus and plot the world
  polys <- refocusWorld(worldLL, xlim, ylim)
  plotMap(polys, xlim, ylim)
  par(oldpar)
})
```

RGB2RYB   *Convert RGB to RYB and RYB to RGB*

Description

Convert RGB (red-green-blue) colours to RYB (red-yellow-blue) colours, and vice versa. Algorithm based on Sugita and Takahashi (2015, 2017)

Usage

```
RGB2RYB(RGBmat)
RYB2RGB(RYBmat)
```

Arguments

RGBmat   numeric – matrix of red-green-blue primary colors by column, where rows are
                individual records of the three primary (RGB) colours. User can specify RGB
                in terms of 0-1 or 0-255; however, the algorith converts the latter to 0-1.

RYBmat   numeric – matrix of red-yellow-blue primary colors by column, where rows are
                individual records of the three primary (RYB) colours. User can specify RYB in
                terms of 0-1 or 0-255; however, the algorith converts the latter to 0-1.

Details

The RYB colour wheel is more commonly used by artists, and provides a more intuitive system
when blending colours – red and yellow makes orange, yellow and blue makes green, blue and red
makes purple. On the RYB colour wheel, red lies opposite green, but on the RGB colour wheel, red
lies opposite cyan.

Value

Matrix of RGB or RYB primary colour intensities, where rows are records and columns are primary colours.

Note

Opposite colours calculated in RYB space (1-RYB) are not always what one expects. For example the colour "purple", RGB {160, 32, 240}, might better be specified as RGB {126, 0, 255} before converting to RYB and inverting.

Author(s)

Rowan Haigh, Program Head – Offshore Rockfish
Pacific Biological Station (PBS), Fisheries & Oceans Canada (DFO), Nanaimo BC
locus opus: Regional Headquarters, Vancouver BC
Last modified `Rd: 2022-07-06`

References

Sugita, J. and Takahashi, T. (2017) Computational RYB Color Model and its Applications. IIEEJ Transactions on Image Electronics and Visual Computing 5(2): 110-122.

Sugita, J. and Takahashi, T. (2015) RYB Color Compositing. International Workshop on Applications in Information Technology, October 8-10, 2015.

See Also

In package **PBSmapping**:
addBubbles

In package **grDevices**:
col2rgb, rgb, rgb2hsv

---

rotatePolys   *Rotate Polygons and Events*

Description

Rotate a PolySet (or an EventData set) clockwise by a specified angle around a fixed point.

Usage

```
rotatePolys(polys, angle=40, centroid=c(500,5700),
   proj.out, zone, xlim=c(-135,-121.5), ylim=c(47,56),
   plot=FALSE, keep.extra=FALSE, ...)

rotateEvents(data, angle=40, centroid=c(500,5700),
   proj.out, zone, plot=FALSE, keep.extra=FALSE, ...)
```

Arguments

polys	PolySet – data frame with fields 'PID', 'SID', 'POS', 'X', 'Y', and attribute 'projection' set to either "LL" or "UTM".
data	EventData – data frame with fields 'EID', 'POS', 'X', 'Y', and attribute 'projection' set to either "LL" or "UTM".
angle	numeric – angle between 0 and 360 degrees for map rotation in a clockwise direction.
centroid	numeric – fixed UTM point, specified in km, around which the map will be rotated. Rotation is performed in UTM space regardless of the input and output projections. If user specifies 'NULL' or 'NA', the centroid of the map will be used.
proj.out	character – desired output projection, where choices are 'LL' or 'UTM'. If 'proj.out' is not specified, the output projection will be the same as the projection of the input object.
zone	numeric – UTM zone used for rotation; if not supplied, the zone is set to 9.
xlim	numeric – two-element vector specifying the limits of X using units (degrees or km) that match the input projection.
ylim	numeric – two-element vector specifying the limits of Y using units (degrees or km) that match the input projection.
plot	logical – if TRUE, plot the results of the rotation.
keep.extra	logical – if TRUE, keep the coordinates of initial and intermediate steps in the rotation process in addition to the final rotated coordinates of the desired projection.
...	dots – additional arguments for 'plotMap' (in 'rotatePolys') or 'plotPoints' (in 'rotateEvents').

Details

Map rotation returns coordinates that are no longer meaningful with respect to the original coordinate system. When displaying rotated maps, the user might wish to turn off axis labels using xaxt="n" and yaxt="n".

Value

Rotated PolySet or EventData set where 'X' and 'Y' are the rotated coordinates in the projection specified by 'proj.out'. The returned object has an attribute list object named 'rotation' that contains:

- angle – angle of clockwise rotation in degrees
- radian – angle of rotation in radians: pi * (-angle)/180
- centroid – fixed point in UTM coordinates (km) around which map is rotated in UTM projection
- R – [rotation matrix](#) (2-dimensional)
- xylim – list object to keep track of 'xlim', 'ylim' and a bounding box 'xybox'.

- `projection` – projection of the rotated PolySet or EventData set
- `zone` – zone of the rotated PolySet or EventData set

When `keep.extra=TRUE`, the returned object will contain additional fields calculated by the rotational algorithm:

- `(X0,Y0)` – original coordinates of the input PolySet | EventData set
- `(uX0,uY0)` – original coordinates converted to UTM (only if original projection is `'LL'`)
- `(aX,aY)` – UTM coordinates adjusted by subtracting the UTM centroid
- `(tX,tY)` – adjusted UTM coordinates transformed by multiplying the rotational matrix
- `(rX,rY)` – rotated UTM coordinates re-centered by adding the UTM centroid

Note:
If `proj.out="UTM"`, the coordinates `c(rX, rY)` are used as the final rotated coordinates. If `proj.out="LL"`, the coordinates `c(rX, rY)` are transformed back into projection `'LL'` as the final rotated coordinates.

Additionally, `'xylim'` in the `'rotation'` list attribute contains intermediary bounding box objects. For instance, if the input PolySet | EventData object has projection `'LL'`, the `'xylim'` object contains:

- `LL` – original `(X,Y)` limits (`'xlim'`, `'ylim'`, `'xybox'`)
- `UTM` – original `(X,Y)` limits transformed to UTM coordinates
- `rot` – rotated UTM `(X,Y)` limits
- `out` – final projection `(X,Y)` limits

Note

The map rotation algorithm is not heavily tested at this time. Report any issues to the package maintainer.

Author(s)

Rowan Haigh, Program Head – Offshore Rockfish
Pacific Biological Station (PBS), Fisheries & Oceans Canada (DFO), Nanaimo BC
locus opus: Institute of Ocean Sciences (IOS), Sidney BC
Last modified `Rd: 2019-03-14`

References

Academo – 2D Rotation about a point

Wikipedia – Rotation matrix

See Also

In package **PBSmapping**:
`as.PolySet` in PolySet, `clipPolys`, `nepacLL`, `plotMap`, `plotPoints`, `refocusWorld`, `surveyData`

---

summary   *Summarize PBS Mapping Objects*

Description

summary method for PBS Mapping classes.

Usage

```
## S3 method for class 'EventData'
summary(object, ...)
## S3 method for class 'LocationSet'
summary(object, ...)
## S3 method for class 'PolyData'
summary(object, ...)
## S3 method for class 'PolySet'
summary(object, ...)
```

Arguments

object	object – a PBSmapping object: EventData, LocationSet, PolyData, or PolySet.
...	dots – further arguments passed to or from other methods.

Details

After creating a list of summary statistics, this function assigns the class `'summary.PBS'` to the output in order to accomplish formatted printing via `print.summary.PBS`.

Value

A list of summary statistics.

Author(s)

[Nicholas M. Boers](#), Staff Software Engineer
Jobber, Edmonton AB
Last modified Rd: `2019-03-14`

See Also

In package **PBSmapping**:
[EventData](#), [LocationSet](#), [PolyData](#), [PolySet](#), `PBSprint`

Examples

```
local(envir=.PBSmapEnv,expr={
  #--- load the data (if using R)
  if (!is.null(version$language) && (version$language=="R"))
    data(surveyData,envir=.PBSmapEnv)
  print(summary(surveyData))
})
```

---

surveyData   *Data: Tow Information from Pacific Ocean Perch Survey*

Description

EventData of Pacific ocean perch (POP) tow information (1966-89).

Usage

```
data(surveyData)
```

Format

Data frame consisting of 9 columns: PID = primary polygon ID, POS = position of each vertex within a given polygon, X = longitude coordinate, Y = latitude coordinate, trip = trip ID, tow = tow number in trip, catch = catch of POP (kg), effort = tow effort (minutes), depth = fishing depth (m), and year = year of survey trip. Attributes: projection = "LL", zone = 9.

Note

In R, the data must be loaded using the data function.

Author(s)

Nicholas M. Boers, Staff Software Engineer
Jobber, Edmonton AB
Last modified Rd: 2008-09-03

Source

The GFBio database, maintained at the Pacific Biological Station (Fisheries and Oceans Canada, Nanaimo, BC V9T 6N7), archives catches and related biological data from commercial groundfish fishing trips and research/assessment cruises off the west coast of British Columbia (BC).

The POP (*Sebastes alutus*) survey data were extracted from GFBio. The data extraction covers bottom trawl surveys that focus primarily on POP biomass estimation: 1966-89 for the central BC coast and 1970-85 for the west coast of Vancouver Island. Additionally, a 1989 cruise along the entire BC coast concentrated on the collection of biological samples. Schnute et al. (2001) provide a more comprehensive history of POP surveys including the subset of data presented here.

References

Schnute, J.T., Haigh, R., Krishka, B.A. and Starr, P. (2001) Pacific ocean perch assessment for the west coast of Canada in 2001. *Canadian Science Advisory Secretariat, Research Document* **2001/138**, 90 pp.

See Also

addPoints, combineEvents, EventData, findPolys, makeGrid, plotPoints.

---

thickenPolys   *Thicken a PolySet of Polygons*

Description

Thicken a PolySet, where each unique (PID, SID) describes a polygon.

Usage

```
thickenPolys (polys, tol = 1, filter = 3, keepOrig = TRUE,
              close = TRUE)
```

Arguments

polys	PolySet to thicken.
tol	tolerance (in kilometres when proj is "LL" and "UTM"; otherwise, same units as polys).
filter	minimum number of vertices per result polygon.
keepOrig	Boolean value; if TRUE, keep the original points in the PolySet.
close	Boolean value; if TRUE, create intermediate vertices between each polygon's last and first vertex, if necessary.

Details

This function thickens each polygon within polys according to the input arguments.

If keepOrig = TRUE, all of the original vertices appear in the result. It calculates the distance between two sequential original vertices, and if that distance exceeds tol, it adds a sufficient number of vertices spaced evenly between the two original vertices so that the distance between vertices no longer exceeds tol. If close = TRUE, it adds intermediate vertices between the last and first vertices when necessary.

If keepOrig = FALSE, only the first vertex of each polygon is guaranteed to appear in the results. From this first vertex, the algorithm walks the polygon summing the distance between vertices. When this cumulative distance exceeds tol, it adds a vertex on the line segment under inspection. After doing so, it resets the distance sum, and walks the polygon from this new vertex. If close = TRUE, it will walk the line segment from the last vertex to the first.

Value

[PolySet](#) containing the thickened data. The function recalculates the POS values for each polygon.

Author(s)

[Nicholas M. Boers](#), Staff Software Engineer
Jobber, Edmonton AB
Last modified Rd: 2013-04-10

See Also

[thinPolys](#).

Examples

```
local(envir=.PBSmapEnv,expr={
  oldpar = par(no.readonly=TRUE)
  #--- load the data (if using R)
  if (!is.null(version$language) && (version$language=="R"))
    data(nepacLL,envir=.PBSmapEnv)
  #--- plot Vancouver Island
  plotMap(nepacLL[nepacLL$PID == 33, ])
  #--- calculate a thickened version using a 30 kilometres tolerance,
  #--- without keeping the original points
  p <- thickenPolys(nepacLL[nepacLL$PID == 33, ], tol = 30, keepOrig = FALSE)
  #--- convert the PolySet to EventData by dropping the PID column and
  #--- renaming POS to EID
  p <- p[-1]; names(p)[1] <- "EID"
  #--- convert the now invalid PolySet into a data frame, and then into
  #--- EventData
  p <- as.EventData(as.data.frame(p), projection="LL")
  #--- plot the results
  addPoints(p, col=2, pch=19)
  par(oldpar)
})
```

---

thinPolys	*Thin a PolySet of Polygons*

Description

Thin a [PolySet](#), where each unique (PID, SID) describes a polygon.

Usage

```
thinPolys (polys, tol = 1, filter = 3)
```

Arguments

polys	[PolySet](#) to thin.
tol	tolerance (in kilometres when proj is ″LL″ and ″UTM″; otherwise, same units as polys).
filter	minimum number of vertices per result polygon.

Details

This function executes the Douglas-Peuker line simplification algorithm on each polygon within polys.

Value

[PolySet](#) containing the thinned data. The function recalculates the POS values for each polygon.

Author(s)

[Nicholas M. Boers](#), Staff Software Engineer
Jobber, Edmonton AB
Last modified Rd: 2013-04-10

See Also

[thickenPolys](#).

Examples

```
local(envir=.PBSmapEnv,expr={
  oldpar = par(no.readonly=TRUE)
  #--- load the data (if using R)
  if (!is.null(version$language) && (version$language=="R"))
    data(nepacLL,envir=.PBSmapEnv)
  #--- plot a thinned version of Vancouver Island (3 km tolerance)
  plotMap(thinPolys(nepacLL[nepacLL$PID == 33, ], tol = 3))
  #--- add the original Vancouver Island in a different line type to
  #--- emphasize the difference
  addPolys(nepacLL[nepacLL$PID == 33, ], border=2, lty=8, density=0)
  par(oldpar)
})
```

---

towData   *Data: Tow Information from Longspine Thornyhead Survey*

Description

[PolyData](#) of tow information for a longspine thornyhead survey (2001).

Usage

```
data(towData)
```

Format

Data frame consisting of 8 columns: `PID` = primary polygon ID, `POS` = position of each vertex within a given polygon, `X` = longitude coordinate, `Y` = latitude coordinate, `depth` = fishing depth (m), `effort` = tow effort (minutes), `distance` = tow track distance (km), `catch` = catch of longspine thornyhead (kg), and `year` = year of survey. Attributes: `projection` = `"LL"`, `zone` = 9.

Note

In R, the data must be loaded using the [data](#) function.

Author(s)

[Nicholas M. Boers](#), Staff Software Engineer
Jobber, Edmonton AB
Last modified `Rd: 2008-09-03`

Source

The GFBio database, maintained at the Pacific Biological Station (Fisheries and Oceans Canada, Nanaimo, BC V9T 6N7), archives catches and related biological data from commercial groundfish fishing trips and research/assessment cruises off the west coast of British Columbia (BC). The longspine thornyhead (*Sebastolobus altivelis*) survey data were extracted from GFBio. Information on the first 45 tows from the 2001 survey (Starr et al. 2002) are included here. Effort is time (minutes) from winch lock-up to winch release.

References

Starr, P.J., Krishka, B.A. and Choromanski, E.M. (2002) Trawl survey for thornyhead biomass estimation off the west coast of Vancouver Island, September 15 - October 2, 2001. *Canadian Technical Report of Fisheries and Aquatic Sciences* **2421**, 60 pp.

See Also

[makeProps](#), [PolyData](#), [towTracks](#).

---

towTracks   *Data: Tow Track Polylines from Longspine Thornyhead Survey*

Description

[PolySet](#) of geo-referenced polyline tow track data from a longspine thornyhead survey (2001).

Usage

```
data(towTracks)
```

Format

Data frame consisting of 4 columns: `PID` = primary polygon ID, `POS` = position of each vertex within a given polyline, `X` = longitude coordinate, and `Y` = latitude coordinate. Attributes: `projection` = `"LL"`, `zone = 9`.

Note

In R, the data must be loaded using the `data` function.

Author(s)

Nicholas M. Boers, Staff Software Engineer
Jobber, Edmonton AB
Last modified `Rd: 2008-09-03`

Source

The longspine thornyhead (*Sebastolobus altivelis*) tow track spatial coordinates are available at the Pacific Biological Station (Fisheries and Oceans Canada, Nanaimo, BC V9T 6N7). The geo-referenced coordinates of the first 45 tows from the 2001 survey (Starr et al. 2002) are included here. Coordinates are recorded once per minute between winch lock-up and winch release.

References

Starr, P.J., Krishka, B.A. and Choromanski, E.M. (2002) Trawl survey for thornyhead biomass estimation off the west coast of Vancouver Island, September 15 - October 2, 2001. *Canadian Technical Report of Fisheries and Aquatic Sciences* **2421**, 60 pp.

See Also

addLines, calcLength, clipLines, plotLines, PolySet, towData.

Index