Package 'inlmisc'

July 23, 2020

Title Miscellaneous Functions for the USGS INL Project Office

Version 0.5.0

Description A collection of functions for creating high-level graphics, performing raster-based analysis, processing MODFLOW-based models, selecting subsets using a genetic algorithm, creating interactive web maps, accessing color palettes, etc. Used to support packages and scripts written by researchers at the United States Geological Survey (USGS) Idaho National Laboratory (INL) Project Office.

Depends R (>= 3.5.0)

Imports checkmate, data.table, GA, graphics, grDevices, htmltools, htmlwidgets, igraph, knitr, leaflet, methods, parallel, raster, rgdal, rgeos, rmarkdown, scales, sp, stats, tinytex, tools, xtable, yaml

Suggests alphahull, dichromat, doParallel, doRNG, gstat, maptools, remotes, roxygen2, testthat

SystemRequirements pandoc - https://pandoc.org/

License CC0

Copyright This software is in the public domain because it contains materials that originally came from the USGS, an agency of the United States Department of Interior.

URL https://github.com/USGS-R/inlmisc

BugReports https://github.com/USGS-R/inlmisc/issues

Encoding UTF-8

LazyData true

LazyDataCompression xz

RoxygenNote 7.1.1

NeedsCompilation yes

Author Jason C. Fisher [aut, cre] (<https://orcid.org/0000-0001-9032-8912>)

Maintainer Jason C. Fisher <jfisher@usgs.gov>

Repository CRAN

Date/Publication 2020-07-23 09:30:02 UTC

R topics documented:

AddColorKey *Add Color Key to Plot*

Description

Add a color key to a plot.

Usage

```
AddColorKey(
  breaks,
  is.categorical = FALSE,
  col = NULL,
  at = NULL,
  labels = TRUE,
  scipen = getOption("scipen", 0),
  explanation = NULL,
  padx = 0.2,
  log = FALSE,
  mai = NULL
)
```

Arguments

breaks	'numeric' vector. Finite breakpoints for the colors: must have one more breakpoint than color and be in increasing order.
is.categorical	'logical' flag. If true, color-key values represent categorical data; otherwise, these data values are assumed continuous.
col	'character' vector. Colors to be used in the plot. This argument requires breaks specification for continuous data. For continuous data there should be one less color than breaks; whereas, categorical data require a color for each category.
at	'numeric' vector. Points at which tick-marks and labels are to be drawn, only applicable for continuous data. The tick marks will be located at the color breaks if the length of at is greater than or equal to one minus the length of breaks. Note that tick-mark labels are omitted where they would abut or overlap previously drawn labels (labels are drawn left to right).
labels	'logical' flag, 'character' vector, 'expression' vector, 'numeric' vector, or 'factor' vector. Can either be a flag specifying whether (numerical) annotations are to be made at the tick marks, or a vector of labels to be placed at the tick points.
scipen	'integer' count. Penalty to be applied when deciding to format numeric values in scientific or fixed notation. Positive values bias towards fixed and negative towards scientific notation: fixed notation will be preferred unless it is more than scipen digits wider. Specify NULL to format all numbers, with the exception of zero, in scientific notation.
explanation	'character' string. Label that describes the data values.

padx	'numeric' number. Inner padding for the left and right margins specified in inches.
log	'logical' flag. Whether the axis is to be logarithmic.
mai	'numeric' vector of length 4. Margin size in inches and of the form `c(bottom,left,top,right)`.

Value

Invisible `NULL`

Author(s)

J.C. Fisher, U.S. Geological Survey, Idaho Water Science Center

See Also

`PlotCrossSection`, `PlotMap`

Examples

```
op <- par(mfrow = c(7, 1), omi = c(1, 1, 1, 1),
          mar = c(2, 3, 2, 3))
AddColorKey(breaks = 0:10,
            explanation = "Example description of data variable.")
AddColorKey(breaks = 0:1000, at = pretty(0:1000))
AddColorKey(breaks = c(0, 1, 2, 4, 8, 16))
breaks <- c(pi * 10^(-5:5))
AddColorKey(breaks = breaks, log = TRUE)
AddColorKey(breaks = breaks,
            at = breaks[as.logical(seq_along(breaks) %% 2)],
            scipen = NULL, log = TRUE)
AddColorKey(is.categorical = TRUE, labels = LETTERS[1:5])
AddColorKey(is.categorical = TRUE,
            col = GetColors(5, scheme = "bright"))
par(op)
```

`AddGradientLegend` *Add Color Gradient Legend to Plot*

Description

Add a continuous color gradient legend strip to a plot.

Usage

```
AddGradientLegend(
  breaks,
  pal = GetColors,
  at = NULL,
  n = 5,
  labels = TRUE,
  scientific = FALSE,
  title = NULL,
  strip.dim = c(2, 8),
  ...
)
```

Arguments

breaks	'numeric' vector. Finite numeric breakpoints for the colors, must be in increasing order.
pal	'function'. Color palette function to be used to assign colors in the legend.
at	'numeric' vector. Points at which tick-marks and labels are to be drawn.
n	'integer' count. Desired number of tick-marks to be drawn. Unused if `at` argument is specified.
labels	'logical' flag or 'character' vector. Can either be a logical value specifying whether annotations are to be made at the tickmarks, or a vector of labels to be placed at the tickpoints.
scientific	'logical' flag. Whether labels should be formatted for scientific notation, see `ToScientific` for details.
title	'character' string. Title to be placed at the top of the legend.
strip.dim	'numeric' vector of length 1 or 2, value is recycled as necessary. Dimensions (width and height) of the color strip, in picas.
...	Additional arguments to be passed to the `GetInsetLocation` function—used to position the legend in the main plot region.

Value

Invisible `NULL`

Author(s)

J.C. Fisher, U.S. Geological Survey, Idaho Water Science Center

See Also

`PlotMap`

Examples

```
plot(NA, xlim = c(0, 100), ylim = c(-10, 10),
     xlab = "x", ylab = "y", xaxs = "i", yaxs = "i")
breaks <- 0:200
AddGradientLegend(breaks, title = "Title", loc = "bottomleft")
AddGradientLegend(breaks, pal = GetColors(scheme = "iridescent"),
                  title = "Title", loc = "bottomleft",
                  inset = c(0.2, 0.1))
AddGradientLegend(breaks, pal = GetColors(scheme = "turbo"),
                  loc = "center", labels = FALSE)
breaks <- seq(0, 2e+06, length.out = 5)
AddGradientLegend(breaks, pal = GetColors(scheme = "discrete rainbow"),
                  scientific = TRUE, strip.dim = c(1, 14),
                  inset = c(0.2, 0.1))
AddGradientLegend(breaks, pal = GetColors(scheme = "YlOrBr"),
                  loc = "topright", inset = 0.1)
```

`AddInsetMap` *Add Inset Map to Plot*

Description

Add an inset map to a plot.

Usage

```
AddInsetMap(
  p,
  col = c("#D8D8D8", "#BFA76F"),
  main.label = list(label = NA, adj = NULL),
  sub.label = list(label = NA, adj = NULL),
  loc = "topright",
  inset = 0.02,
  width = NULL,
  e = graphics::par("usr"),
  bty = c("o", "n"),
  feature = NULL
)
```

Arguments

`p`	'SpatialPolygons'. Polygon describing the large map.
`col`	'character' vector of length 2. Colors for filling the large map polygon `p` and the smaller plot extent rectangle.
`main.label`	'list'. List with components `label` and `adj`. The text label and position (x and y adjustment of the label) for the large map, respectively.

sub.label	'list'. Identical to the main.label argument but for the plot extent rectangle.
loc	'character' string. Position of the inset map in the main plot region; see GetInsetLocation function for keyword descriptions.
inset	'numeric' vector of length 1 or 2, value is recycled as necessary. Inset distance(s) from the margins as a fraction of the main plot region. Defaults to 2 percent of the axis range.
width	'numeric' number. Width of the inset map in inches.
e	'numeric' vector of length 4. Extent of the smaller axis-aligned rectangle (relative to the larger map polygon). Defaults to the user coordinate extent of the main plot region.
bty	'character' string. Type of box to be drawn about the inset map. A value of "o" (the default) results in a box and a value of "n" suppresses the box.
feature	'list'. One or more spatial objects, along with style arguments, to add to the inset map. Each list element is a 'list'-class object that contains the following components: the first component is the name of the plotting function; the second component is the object to be plotted; and the remaining components are reserved for arguments to be passed to the function.

Value

Invisible NULL

Author(s)

J.C. Fisher, U.S. Geological Survey, Idaho Water Science Center

See Also

PlotMap

Examples

```
file <- system.file("extdata/county.geojson",
                    package = "inlmisc")[1]
county <- rgdal::readOGR(file)
ext <- c(-113.4005, -112.2764, 43.30, 44.11)
PlotMap(county, xlim = ext[1:2], ylim = ext[3:4],
        dms.tick = TRUE)
sp::plot(county, add = TRUE)
AddInsetMap(county, width = 2,
            main.label = list("IDAHO", "adj" = c(0, -10)),
            sub.label = list("Map area", "adj" = c(0, -4)),
            loc = "topright")
```

`AddIntervals` *Add Interval Symbols to Plot*

Description

Add interval symbols (also known as error bars) to plots.

Usage

```
AddIntervals(
  x,
  y0,
  y1,
  hin = NULL,
  col = "black",
  lty = 1,
  lwd = 0.7,
  cex = 1,
  xpd = FALSE,
  ...,
  nondetects = NULL
)
```

Arguments

`x`	'numeric' or 'Date' vector. *x* coordinate of interval symbols.
`y0`	'numeric' vector. *y* coordinate of points from which to draw.
`y1`	'numeric' vector. *y* coordinate of points to which to draw.
`hin`	'numeric' number. Horizontal length of an interval head, in inches.
`col, lty, lwd, cex, xpd`	graphical parameters; see `par` for details. `NA` values in `col` cause the interval to be omitted.
`...`	Additional graphical parameters to the `points` function.
`nondetects`	'list'. Overrides graphical parameters used for left- and right-censored data. Passed arguments include `col`, `lty`, and `lwd`.

Details

For each observation `i`, the data type is identified using `(y0[i],Inf)` for right-censored, `y0[i] = y1[i]` for exact, and `(-Inf,y1[i])` for left-censored, and `(y0[i],y1[i])` for interval-censored. Where infinity may be represented with either `Inf` or `NA`.

Value

Invisible `NULL`

Author(s)

J.C. Fisher, U.S. Geological Survey, Idaho Water Science Center

Examples

```
set.seed(1)
x <- stats::runif(12)
y <- stats::rnorm(12)
plot(x, y)
dy <- sort.int(y) / 5
AddIntervals(x, y - dy, y + dy, col = "red", xpd = TRUE)

n <- 50
x <- sort.int(stats::runif(n, max = 100))
y1 <- y0 <- stats::runif(n, max = 100)
y1[sample.int(n, 5)] <- stats::runif(5, max = 100)
y0[sample.int(n, 5)] <- -Inf
y1[sample.int(n, 5)] <-  Inf
ylim <- range(pretty(c(y0, y1)))
plot(NA, xlim = range(x), ylim = ylim, xlab = "x", ylab = "y")
AddIntervals(x, y0, y1, col = "blue", xpd = TRUE,
             nondetects = list("col" = "red", "lty" = 2))
print(cbind(x, y0, y1))
```

AddNorthArrow *Add North Arrow to Plot*

Description

Add a north arrow aligned to true north to a plot.

Usage

```
AddNorthArrow(crs = sp::CRS(), len = 0.05, lab = "N", rotate = 0, ...)
```

Arguments

crs	'CRS', 'Raster*', or 'Spatial'. Coordinate reference system (CRS), or any object with a CRS attribute that can be extracted using the crs function. If missing (the default) the north arrow is point to the top of the plot unless the rotate argument is specified.
len	'numeric' number. Arrow length expressed as a fraction of the plot height, by default is 5-percent.
lab	'character' string. North label, by default is "N".
rotate	'numeric' number. Arrow offset-rotation in degrees, where positive values are taken to be clockwise.
...	Additional arguments to be passed to the GetInsetLocation function—used to position the north arrow in the main plot region.

Value

Invisible NULL

Author(s)

J.C. Fisher, U.S. Geological Survey, Idaho Water Science Center

See Also

PlotMap

Examples

```
m <- datasets::volcano
m <- m[nrow(m):1, ncol(m):1]
x <- seq(from = 2667405, length.out = ncol(m), by = 10)
y <- seq(from = 6478705, length.out = nrow(m), by = 10)
r <- raster::raster(m, xmn = min(x), xmx = max(x),
                    ymn = min(y), ymx = max(y),
                    crs = "+init=epsg:27200")
PlotMap(r, pal = GetColors(scheme = "DEM screen"))
AddNorthArrow(raster::crs(r), loc = "center")
AddNorthArrow(raster::crs(r), inset = 0.1)
AddNorthArrow(raster::crs(r), loc = "topleft", inset = 0.1)
```

AddPoints *Add Points to Plot*

Description

Add point symbols to a plot. Proportional circle symbols may be used to represent point data, where symbol area varies in proportion to an attribute variable.

Usage

```
AddPoints(
  x,
  y = NULL,
  z = NULL,
  zcol = 1,
  crs = NULL,
  xlim = NULL,
  ylim = NULL,
  zlim = NULL,
  inches = c(0, 0.2),
  scaling = c("perceptual", "mathematical", "radius"),
  bg = "#1F1F1FCB",
```

```
  bg.neg = NULL,
  fg = NA,
  lwd = 0.7,
  cex = 0.7,
  format = NULL,
  legend.loc = "topright",
  inset = 0.02,
  bty = c("o", "n"),
  breaks = NULL,
  break.labels = NULL,
  quantile.breaks = FALSE,
  make.intervals = FALSE,
  title = NULL,
  subtitle = NULL,
  draw.legend = TRUE,
  draw.points = FALSE,
  add = TRUE,
  ...
)
```

Arguments

x, y	'numeric' vector or 'SpatialPoints*'. *x* and *y* coordinates for the centers of the circle symbols. If numeric, can be specified in any way which is accepted by xy.coords.
z	'numeric' vector, 'integer' vector, or 'factor'. Attribute variable. For objects of class factor, a fixed radius is used for circle symbols, see inches argument description.
zcol	'integer' count or 'character' string. Attribute name or column number to extract from if x is of class 'SpatialGridDataFrame'.
crs	'character' string or 'CRS'. Coordinate reference system arguments
xlim	'numeric' vector of length 2. *x* limits for the plot.
ylim	'numeric' vector of length 2. *y* limits for the plot.
zlim	'numeric' vector of length 2. *z* limits for the plot.
inches	'numeric' vector of length 2. Radii limits for the drawn circle symbol. Alternatively, a single number can be given resulting in a fixed radius being used for all circle symbols; this overrides proportional circles and the function behaves like the points function.
scaling	'character' string. Selects the scaling algorithm to use for symbol mapping. Specify "perceptual" or "mathematical" for proportional scaling (Tanimura and others, 2006), or "radius" for scaling symbol size to radius (usually a bad idea).
bg	'character' vector or 'function'. Fill color(s) for circle symbols. A color palette also may be specified.
bg.neg	'character' vector or 'function'. Fill color(s) for circle symbols corresponding to negative z values. A color palette also may be specified. For circle symbols corresponding to positive z values, the bg argument is used for color(s).

fg	'character' string. Outer-line color for circle symbols. Specify a value of NA to remove the symbols outer line, and NULL to match the outer-line color with the symbols fill color.
lwd	'numeric' number. Line width for drawing circle symbols.
cex	'numeric' number. Character expansion factor for legend labels.
format	'character' string. Formatting for legend values, see formatC for options.
legend.loc	'character' string. Position of the legend in the main plot region; see GetInsetLocation function for keyword descriptions.
inset	'numeric' number. Inset distance of the legend from the margins as a fraction of the main plot region. Defaults to 2 percent of the axis range.
bty	'character' string. Type of box to be drawn about the legend. A value of "o" (the default) results in a box and a value of "n" suppresses the box.
breaks	'numeric' vector. Finite breakpoints for the legend circle symbols.
break.labels	'character' vector. Break labels with length equal to breaks.
quantile.breaks	'logical' flag. If true, breaks are set to the sample quantiles of z.
make.intervals	'logical' flag. If true, represent z within intervals. See findInterval function for details. Unused if quantile.breaks is true.
title	'character' string. Main title to be placed at the top of the legend.
subtitle	'character' string. Legend subtitle to be placed below the main title.
draw.legend	'logical' flag. If true, a legend is drawn.
draw.points	'logical' flag. If true, the circle symbols are drawn.
add	'logical' flag. If true, circle symbols (and an optional legend) are added to an existing plot.
...	Graphics parameters to be passed to PlotMap. Unused if add = TRUE.

Value

Invisible NULL

Note

To avoid overplotting, circle symbols are drawn in order of decreasing radius.

Author(s)

J.C. Fisher, U.S. Geological Survey, Idaho Water Science Center

References

Tanimura, S., Kuroiwa, C., and Mizota, T., 2006, Proportional Symbol Mapping in R: Journal of Statistical Software, v. 15, no. 5, 7 p.

Examples

```
set.seed(2)
n <- 50L
x <- cbind(runif(n, 1, 10), runif(n, 1, 500))
z <- runif(n, 0, 1000)
z[sample.int(n, 2)] <- 0
AddPoints(x, z = z, fg = "#00000080", lwd = 0.5, title = "Title",
          subtitle = "Subtitle", add = FALSE)

idxs <- sample.int(n, floor(n / 2))
z[idxs] <- -z[idxs]
AddPoints(x, z = z, bg.neg = "#2A8FBDCB",
          breaks = pretty(z, n = 8),
          legend.loc = "bottomleft", add = FALSE)

Pal1 <- colorRampPalette(c("#CA0020CB", "#F4A582CB"), alpha = TRUE)
Pal2 <- colorRampPalette(c("#0571B0CB", "#92C5DECB"), alpha = TRUE)
AddPoints(x, z = z, bg = Pal1, bg.neg = Pal2, add = FALSE)

AddPoints(x, z = z, bg = Pal1, bg.neg = Pal2, add = FALSE,
          make.intervals = TRUE)

AddPoints(x, z = z, bg = Pal1, bg.neg = Pal2, add = FALSE,
          make.intervals = TRUE, inches = 0.1)

AddPoints(x, z = abs(z), title = "Quantiles", bg = topo.colors,
          quantile.breaks = TRUE, add = FALSE)

z <- as.factor(rep(c("dog", "cat", "ant", "pig", "bat"),
                   length.out = n))
bg <- GetColors(nlevels(z), scheme = "bright", alpha = 0.8)
AddPoints(x, z = z, bg = bg, add = FALSE)

AddPoints(x, draw.legend = FALSE, add = FALSE)
```

AddScaleBar *Add Scale Bar to Plot*

Description

Add a scale bar (also known as a rake scale) to a plot.

Usage

```
AddScaleBar(
  unit = NULL,
  conv.fact = NULL,
  vert.exag = NULL,
```

```
  longlat = FALSE,
  loc = "bottomleft",
  ...
)
```

Arguments

unit	'character' vector of length 1 or 2, value is recycled as necessary. Label(s) describing the unit of measurement of scale distances, such as "METERS".
conv.fact	'numeric' vector of length 1 or 2, value is recycled as necessary. Conversion factor(s) for changing the unit of measurement for scale distances. For example, if user coordinates of the plotting region are in meters, specify `3.28084` to display scale distances in feet. A dual-unit scale bar is created by specifying a second conversion factor.
vert.exag	'logical' flag, 'numeric' vector, or 'character' vector. Either a logical value indicating whether to include a vertical exaggeration label; or a custom *y/x* aspect ratio to include in this label.
longlat	'logical' flag. Whether user coordinates of the plotting region are in longitude and latitude; if true, scale distances are in kilometers. Note that scale distances are calculated at the maps latitude midpoint using the Great Circle distance (WGS84 ellipsoid) method.
loc	'character' string. Position of the scale bar in the main plot region; see `GetInsetLocation` function for keyword descriptions.
...	Additional arguments to be passed to the `GetInsetLocation` function—used to position the scale bar in the main plot region.

Value

Invisible `NULL`

Author(s)

J.C. Fisher, U.S. Geological Survey, Idaho Water Science Center

See Also

`PlotMap`, `PlotCrossSection`

Examples

```
plot(-100:100, -100:100, type = "n", xlab = "x in meters",
     ylab = "y in meters", asp = 2)
AddScaleBar()
AddScaleBar(loc = "center")
AddScaleBar(unit = "METERS", loc = "topleft", padin = 0.2)
AddScaleBar(unit = c("METERS", "FEET"), conv.fact = c(1, 3.28084),
            loc = "topright", padin = c(0.5, 0))
AddScaleBar(unit = c("METERS", "FEET"), conv.fact = c(1, 3.28084),
            vert.exag = TRUE, loc = "bottomright", inset = 0.1)
```

```
plot(c(-38.31, -35.5), c(40.96, 37.5), type = "n",
     xlab = "longitude", ylab = "latitude")
AddScaleBar(unit = "KILOMETERS", longlat = TRUE)
AddScaleBar(unit = "MILES", conv.fact = 0.621371, longlat = TRUE,
            loc = "topright", padin = c(0.4, 0))
AddScaleBar(unit = c("KILOMETERS", "MILES"),
            conv.fact = c(1, 0.621371), longlat = TRUE,
            loc = "topleft", inset = 0.05)
```

AddWebMapElements *Add Elements to Web Map*

Description

Augment a Leaflet web map with additional elements. The AddHomeButton function adds a button that zooms to the initial map extent. The AddClusterButton function adds a button that toggles marker clusters on and off. The AddSearchButton function adds a control that may be used to search markers/features location by property. And the AddCircleLegend function adds a map legend.

Usage

```
AddHomeButton(map, extent = NULL, position = "topleft")

AddClusterButton(map, clusterId, position = "topleft")

AddSearchButton(
  map,
  group,
  propertyName = "label",
  zoom = NULL,
  textPlaceholder = "Search...",
  openPopup = FALSE,
  position = "topleft"
)

AddLegend(
  map,
  labels,
  colors,
  radius,
  opacity = 0.5,
  symbol = c("square", "circle"),
  title = "EXPLANATION",
  position = "topright"
)
```

Arguments

map	'leaflet'. Map widget object
extent	'Spatial*', 'Raster*', 'Extent', 'matrix', or 'numeric' vector. Extent object (or object from which an raster::extent object can be extracted/created) representing a rectangular geographical area on the map. The extent must be specified in the coordinate reference system (CRS) of the web map, usually in latitude and longitude using WGS 84 (also known as EPSG:4326). By default, the extent object is read from the map widget.
position	'character' string. Position of the button on the web map. Possible values are "topleft", "topright", "bottomleft", and "bottomright".
clusterId	'character' string. Identification for the marker cluster layer.
group	'character' string. Name of the group whose features will be searched.
propertyName	'character' string. Property name used to describe markers, such as, "label" and "popup".
zoom	'integer' count. Zoom level for move to location after marker found in search.
textPlaceholder	'character' string. Message to show in search element.
openPopup	'logical' flag. Whether to open the marker popup associated with the searched for marker.
labels	'character' vector. Labels in the legend.
colors	'character' vector. HTML colors corresponding to labels.
radius	'numeric' number. Border radius of symbols in the legend, in pixels.
opacity	'numeric' number. Opacity of symbols in the legend, from 0 to 1.
symbol	'character' string. Symbol type in the legend, either "square" or "circle".
title	'character' string. Legend title

Value

An object of class 'leaflet'. A new map object with added element.

Author(s)

J.C. Fisher, U.S. Geological Survey, Idaho Water Science Center

See Also

CreateWebMap

Examples

```
city <- rgdal::readOGR(system.file("extdata/city.geojson",
                       package = "inlmisc")[1])
opt <- leaflet::markerClusterOptions(showCoverageOnHover = FALSE)
map <- CreateWebMap("Topo")
map <- leaflet::addMarkers(map, label = ~name, popup = ~name,
```

```
                          clusterOptions = opt,
                          clusterId = "cluster",
                          group = "marker", data = city)
map <- AddHomeButton(map)
map <- AddClusterButton(map, clusterId = "cluster")
map <- AddSearchButton(map, group = "marker", zoom = 15,
                       textPlaceholder = "Search city names...")
map

labels <- c("Non-capital", "Capital")
colors <- c("green", "red")
fillColor <- colors[(city@data$capital > 0) + 1]
map <- CreateWebMap("Topo")
map <- leaflet::addCircleMarkers(map, radius = 6, color = "white",
                                 weight = 1, opacity = 1,
                                 fillColor = fillColor,
                                 fillOpacity = 1, fill = TRUE,
                                 data = city)
map <- AddLegend(map, labels = labels, colors = colors, radius = 5,
                 opacity = 1, symbol = "circle")
map
```

`BuildVignettes` *Build Package Vignettes*

Description

Build package vignettes from their source files using the `buildVignettes` function. Writes the PDF and (or) HTML documents of the package vignettes, and their executable code files.

Usage

```
BuildVignettes(
  dir = getwd(),
  doc = file.path(dir, "inst/doc"),
  gs_quality = c("ebook", "printer", "screen", "none"),
  clean = TRUE,
  quiet = TRUE
)
```

Arguments

`dir`	'character' string. Path to a package's root source directory, by default the working directory. Its subdirectory 'vignettes' is searched for vignette source files.
`doc`	'character' string. Path to write the vignette output files, by default 'inst/doc' under the working directory.

gs_quality	'character' string. Quality of compacted PDF files, the options are "ebook" (150 dpi, default), "printer" (300 dpi), "screen" (72 dpi), and "none" (no compression). See compactPDF function for details.
clean	'logical' flag. Whether to remove all intermediate files generated by the build.
quiet	'logical' flag. Whether to suppress most output.

Value

Invisible NULL

Author(s)

J.C. Fisher, U.S. Geological Survey, Idaho Water Science Center

---

BumpDisconnectCells *Adjust Vertically Disconnected Cells*

Description

Given upper and lower surfaces (raster layers) of a three-dimensional (3D) model layer, this function incrementally decreases lower cell values until a minimum vertical overlap between adjacent model cells is satisfied.

Usage

```
BumpDisconnectCells(rs, min.overlap = 2, bump.by = 0.1, max.itr = 10000)
```

Arguments

rs	'Raster*'. Collection of two raster layers, the first and second layers represent the upper and lower surface of a 3D model layer.
min.overlap	'numeric' number. Minimum vertical overlap between horizontally adjacent model cells.
bump.by	'numeric' number. Amount to decrease a cell value by during each iteration of the algorithm.
max.itr	'integer' count. Maximum number of iterations.

Details

During each iteration of the algorithm: (1) Cells are identified that violate the minimum vertical overlap between adjacent cells; that is, the bottom of cell *i* is greater than or equal to the top of an adjacent cell *j* minus the minimum overlap specified by the min.overlap argument. (2) For cells violating the minimum vertical overlap, lower raster layer (rs[[2]]) values are decreased by the value specified in the bump.by argument.

Value

An object of class 'RasterLayer' that can be added to `rs[[2]]` to ensure connectivity between model layer cells. Cell values in the returned raster grid represent vertical adjustments.

Author(s)

J.C. Fisher, U.S. Geological Survey, Idaho Water Science Center

Examples

```
set.seed(0)
r_top <- raster::raster(ncols = 10, nrows = 10)
r_bot <- raster::raster(ncols = 10, nrows = 10)
r_top[] <- rnorm(raster::ncell(r_top), mean = 12)
r_bot[] <- rnorm(raster::ncell(r_bot), mean = 10)
rs <- raster::stack(r_top, r_bot)
r <- BumpDisconnectCells(rs, min.overlap = 0.1)
raster::plot(r, col = GetColors(255, reverse = TRUE))
summary(r[])

r_bot_new <- r_bot + r
```

`BumpRiverStage` *Adjust Implausible River Stage*

Description

Decrease stage values in river cells if they are implausible with respect to water always flowing downhill.

Usage

```
BumpRiverStage(r, outlets, min.drop = 1e-06)
```

Arguments

`r`	'RasterLayer'. Numeric cell values representing river stages.
`outlets`	'SpatialPoints*', 'SpatialLines*', 'SpatialPolygons*' or 'Extent'. Designates the location of discharge outlets. The `rasterize` function is used to locate outlet cells in the raster grid `r`.
`min.drop`	'numeric' number. Minimum drop in stage between adjacent river cells.

Details

The Lee algorithm (Lee, 1961) is used to identify flow paths among the modeled river cells. An analysis of river cell stage values along a flow path identifies any problematic cells that are obstructing downhill surface-water flow. Stage values for these problematic cells are then lowered to an acceptable elevation.

Value

An object of class 'RasterLayer' with cell values representing the vertical change in stream stage. These changes can be added to `r` to ensure that water always flows downhill.

Author(s)

J.C. Fisher, U.S. Geological Survey, Idaho Water Science Center

References

Lee, C.Y., 1961, An algorithm for path connections and its applications: IRE Transactions on Electronic Computers, v. EC-10, no. 2, p. 346–365.

`CreateWebMap` *Create a Web Map Using TNM Services*

Description

Create a Leaflet map widget with base maps offered through The National Map (TNM). Information about the content of these base maps can be found within the TNM Base Maps document.

Usage

```
CreateWebMap(maps, ..., collapsed = TRUE, service = c("rest", "wms"))
```

Arguments

`maps`	'character' vector. TNM base maps to include in the web map. Possible maps include `"Topo"`, `"Imagery"`, `"Imagery Topo"`, `"Hydrography"`, `"Hill Shade"`, and `"Blank"`. All base maps are included by default.
`...`	Arguments to be passed to the `leaflet` function.
`collapsed`	'logical' flag. Whether the layers control should be rendered as an icon that expands when hovered over.
`service`	'character' string. Mapping services for accessing TNM base-map tiles. Select `"rest"` for representational state transfer services (the default) and `"wms"` for web map services.

Details

Map service endpoints are offered through TNM with no use restrictions. However, map content is limited to the United States and territories. This function integrates TNM endpoint services within an interactive web map using Leaflet for R.

Value

An object of class 'leaflet', a hypertext markup language (HTML) widget object. See example for instructions on how to add additional graphic layers (such as points, lines, and polygons) to the map widget. Graphic layers added to the web map must be in latitude and longitude using WGS 84 (also known as EPSG:4326).

Author(s)

J.C. Fisher, U.S. Geological Survey, Idaho Water Science Center

See Also

AddWebMapElements

Examples

```
map <- CreateWebMap(collapsed = FALSE)
ll <- rbind(c(-112.049705, 43.517810),
            c(-122.171257, 37.456526),
            c( -77.367458, 38.947206),
            c(-149.803565, 61.187905),
            c( -80.248344, 26.080860))
map <- leaflet::addMarkers(map, ll[, 1], ll[, 2])
map
```

ExportRasterStack *Export a Raster Stack*

Description

Write a raster-stack, a collection of raster layers, to local directories using multiple file formats.

Usage

```
ExportRasterStack(rs, path, zip = "", col = NULL)
```

Arguments

rs	'RasterStack' or 'RasterBrick'. Collection of RasterLayer objects with the same extent and resolution.
path	'character' string. Path name to write raster stack.
zip	'character' string. If there is no zip program on your path (on windows), you can supply the full path to a 'zip.exe' here, in order to make a KMZ file.
col	'character' vector. Color names

Details

Five local directories are created under path and named after their intended file formats: Comma-Separated Values ('csv'), Portable Network Graphics ('png'), Georeferenced TIFF ('tif'), R Data ('rda'), and Keyhole Markup Language ('kml'). For its reference system, 'kml' uses geographic coordinates: longitude and latitude components as defined by the World Geodetic System of 1984. Therefore, the conversion of gridded data between cartographic projections may introduce a new source of error.

Value

Invisible NULL

Note

If the zip program is unavailable on windows, install it by downloading the latest binary version from the Info-ZIP website; select one of the given FTP locations, enter directory 'win32', download 'zip300xn.zip', and extract.

Author(s)

J.C. Fisher, U.S. Geological Survey, Idaho Water Science Center

Examples

```
## Not run:
rs <- raster::stack(system.file("external/rlogo.grd",
                                package = "raster"))
print(rs)
path <- file.path(getwd(), "rlogo")
dir.create(path)
ExportRasterStack(rs, path)
list.files(normalizePath(path, winslash = "/"),
           full.name = TRUE, recursive = TRUE,
           include.dirs = TRUE)

unlink(path, recursive = TRUE)

## End(Not run)
```

ExtractAlongTransect *Extract Raster Values Along a Transect Line*

Description

Extract values from raster layer(s) along a user defined transect line.

Usage

```
ExtractAlongTransect(transect, r)
```

Arguments

transect	'SpatialPoints' or 'SpatialLines'. Transect line or its vertices.
r	'RasterLayer', 'RasterStack', or 'RasterBrick'. Raster layer(s)

Details

The transect line is described using a simple polygonal chain. The transect line and raster layer(s) must be specified in a coordinate reference system.

Value

A 'list' with components of class 'SpatialPointsDataFrame'. These components represent continuous piecewise line segments along the transect. The following variables are specified for each coordinate point in the line segment:

dist distance along the transect line.

2, . . . , n extracted value for each raster layer in r, where column names match their respective raster layer name.

Author(s)

J.C. Fisher, U.S. Geological Survey, Idaho Water Science Center

See Also

PlotCrossSection

Examples

```
coords <- rbind(c(-100, -90), c(80, 90), c(80, 0), c(40, -40))
crs <- sp::CRS("+init=epsg:4326")
transect <- sp::SpatialPoints(coords, proj4string = crs)
r <- raster::raster(nrows = 10, ncols = 10,
                    ymn = -80, ymx = 80, crs = crs)
names(r) <- "value"
set.seed(0)
r[] <- runif(raster::ncell(r))
r[4, 6] <- NA
PlotMap(r)
l <- sp::Lines(list(sp::Line(coords)), ID = "Transect")
lines(sp::SpatialLines(list(l), proj4string = crs))
points(transect, pch = 19)
segs <- ExtractAlongTransect(transect, r)
for (i in seq_along(segs)) points(segs[[i]])

dev.new()
```

```
xlab <- "Distance along transect"
ylab <- "Raster value"
xlim <- range(vapply(segs, function(i) {
  range(i@data[, "dist"])
}, c(0, 0)))
ylim <- range(vapply(segs, function(i) {
  range(i@data[, "value"], na.rm = TRUE)
}, c(0, 0)))
PlotGraph(NA, xlab = xlab, ylab = ylab,
          xlim = xlim, ylim = ylim, type = "n")
cols <- GetColors(length(segs), scheme = "bright")
for (i in seq_along(segs))
  lines(segs[[i]]@data[, c("dist", "value")],
        col = cols[i], lwd = 2)
coords <- sp::coordinates(transect)
n <- length(transect)
d <- cumsum(c(0, as.matrix(dist((coords)))[cbind(1:(n - 1), 2:n)]))
abline(v = d, lty = 2)
mtext(sprintf("(%d, %d)", coords[1, 1], coords[1, 2]),
      line = -1, adj = 0, cex = 0.7)
mtext(sprintf("(%d, %d)", coords[n, 1], coords[n, 2]),
      line = -1, adj = 1, cex = 0.7)

graphics.off()
```

`FindOptimalSubset` *Find Optimal Subset Using a GA*

Description

Find optimal subset of a fixed size `k` from a finite sequence of length `n`. A distributed multiple-population genetic algorithm (GA) is used to do subset selection based on the maximization of a user-supplied fitness function.

Usage

```
FindOptimalSubset(
  n,
  k,
  Fitness,
  ...,
  popSize = 100,
  numIslands = 4,
  migrationRate = 0.1,
  migrationInterval = 10,
  pcrossover = 0.8,
  pmutation = 0.1,
  elitism = max(1, round(popSize/numIslands * 0.05)),
```

```
  maxiter = 1000,
  run = maxiter,
  suggestions = NULL,
  parallel = TRUE,
  monitor = NULL,
  seed = NULL
)
```

Arguments

n	'integer' count. Maximum permissible index, that is, the length of the finite sequence (1:n). The GA chooses a subset from this sequence.
k	'integer' count. Number of indices to choose, that is, the fixed size of the subset.
Fitness	'function'. Fitness function, also known as the objective function, is any allowable R function which takes as its first argument the binary string representing a potential solution. And as its second argument the maximum permissible index, n. Use the DecodeChromosome(string,n) command to decode the binary string. The fitness function returns a single numerical value describing its fitness. Recall that the GA searches for a maximum fitness value.
...	Additional arguments to be passed to the fitness function.
popSize	'integer' count. Population size that is distributed evenly between islands.
numIslands	'integer' count. Number of islands
migrationRate	'numeric' number. Proportion of individuals that should migrate between islands.
migrationInterval	'integer' count. Number of generations at which exchange of individuals (or migration) takes place. This interval between migrations is called an *epoch*.
pcrossover	'numeric' number. Probability of crossover between pairs of chromosomes.
pmutation	'numeric' number. Probability of mutation in a parent chromosome.
elitism	'integer' count. Number of chromosomes to survive into the next generation. Defaults to 5-percent of the island population.
maxiter	'integer' count. Maximum number of generations to run on each island before the GA search is halted.
run	'integer' count. Number of consecutive generations without any improvement in the "best" fitness value before the GA is stopped.
suggestions	integer 'matrix'. Integer chromosomes to be included in the initial population. See returned solution component for a suggested value for this argument.
parallel	'logical' flag or 'integer' count. Whether to use parallel computing. This argument can also be used to specify the number of cores to employ; by default, this is the number of physical CPUs/cores. The **parallel** and **doParallel** packages must be installed for parallel computing to work.
monitor	'function'. Function that takes as input the current state of the gaisl-class object, and is run at each epoch of the islands GA search.
seed	'integer' count. Random number generator state for random number generation, used to replicate the results. The **doRNG** package must be installed if using parallel computing.

Details

The fitness function (see `Fitness` argument) is solved using the `gaisl` function in the **GA** package (Scrucca, 2013, 2016). The function implements an islands evolution model (first proposed by Cohoon and others, 1987). to maximize a fitness function using islands parallel genetic algorithms (ISLPGAs) (Luke, 2013, p. 103-104; Scrucca, 2016, p. 197-200). Independent GAs are configured to use integer chromosomes represented with a binary codification, linear-rank selection, uniform crossover, and uniform mutation.

Value

A 'list' with components:

`call` original call which can be used for later re-use.

`solution` a 'matrix' representation of the best solution found. Each row represents a unique solution giving the best fitness at the final generation. More than one row indicates a non-unique solution. The number of columns is equal to the subset size `k`.

`ga_output` output from the ISLPGAs, see `gaisl-class` for format description.

`ga_time` time required to run the ISLPGAs, see `system.time` for details.

Author(s)

J.C. Fisher, U.S. Geological Survey, Idaho Water Science Center

References

Cohoon, J.P., Hegde, S.U., Martin, W.N., and Richards, D., 1987, Punctuated Equilibria: A Parallel Genetic Algorithm, in Genetic Algorithms and their Applications: Proceedings of the Second International Conference on Genetic Algorithms, Grefenstette, J.J., Lawrence Earlbaum Associates, p. 155-161.

Luke, Sean, 2015, Essentials of metaheuristics (2nd ed.): Lulu, 263 p., available for free at https://cs.gmu.edu/~sean/book/metaheuristics/.

Scrucca, Luca, 2013, GA: A Package for Genetic Algorithms in R: Journal of Statistical Software, v. 53, no. 4, p. 1-37, http://dx.doi.org/10.18637/jss.v053.i04.

Scrucca, Luca, 2017, On some extensions to GA package: hybrid optimisation, parallelisation and islands evolution: The R Journal, v. 9, no. 1, p. 187-206, https://journal.r-project.org/archive/2017/RJ-2017-008.

Examples

```
# Problem: Choose the 4 smallest numbers from a list
#          of 100 values generated from a standard
#          uniform distribution.
k <- 4
n <- 100
seed <- 123
set.seed(seed); numbers <- sort.int(runif(n))
Fitness <- function(string, n, numbers) {
  idxs <- DecodeChromosome(string, n)
```

```
  -1 * sum(numbers[idxs])
}
## Not run:
out <- FindOptimalSubset(n, k, Fitness, numbers, run = 10,
                         monitor = GA::gaislMonitor, seed = seed)
plot(out[["ga_output"]])
summary(out[["ga_output"]])
print(out[["solution"]])
print(out[["ga_output"]]@fitnessValue)

## End(Not run)
```

`FormatPval` *Format P Values*

Description

Format *p*-values for pretty printing.

Usage

```
FormatPval(
  x,
  digits = max(1, getOption("digits") - 2),
  eps = .Machine$double.eps,
  na.form = "NA",
  scientific = NA
)
```

Arguments

`x`	'numeric' vector. *p*-values
`digits`	'integer' count. Number of significant digits to be used.
`eps`	'numeric' number. Numerical tolerance, values less than `eps` are formatted as `"< [eps]"`.
`na.form`	'character' string. Value used for missing values.
`scientific`	'logical' flag. Whether values should be encoded in scientific format using LaTeX notation. A missing value lets R decide whether fixed or scientific notation is used.

Value

A 'character' vector of formatted *p*-values.

Author(s)

J.C. Fisher, U.S. Geological Survey, Idaho Water Science Center

See Also

ToScientific

Examples

```
x <- c(stats::runif(5), pi^-100, NA)
FormatPval(x)
format.pval(x)

x <- c(0.1, 0.0001, 1e-27)
FormatPval(x, scientific = TRUE)
FormatPval(x, digits = 3, eps = 0.001)
```

GetColors *Get Palette Colors*

Description

Create a vector of n colors from qualitative, diverging, and sequential color schemes.

Usage

```
GetColors(
  n,
  scheme = "smooth rainbow",
  alpha = NULL,
  stops = c(0, 1),
  bias = 1,
  reverse = FALSE,
  blind = NULL,
  gray = FALSE,
  ...
)
```

Arguments

n	'integer' count. Number of colors to be in the palette. The maximum number of colors in a generated palette is dependent on the specified color scheme, see 'Details' section for maximum values.
scheme	'character' string. Name of color scheme, see 'Details' section for scheme descriptions. Argument choices may be abbreviated as long as there is no ambiguity.
alpha	'numeric' number. Alpha transparency, values range from 0 (fully transparent) to 1 (fully opaque). Specify as NULL to exclude the alpha channel value from colors.

`stops` 'numeric' vector of length 2. Color stops defined by interval endpoints (between 0 and 1) and used to select a subset of the color palette. Only suitable for schemes that allow for color interpolations.

`bias` 'numeric' number. Interpolation bias where larger values result in more widely spaced colors at the high end.

`reverse` 'logical' flag. Whether to reverse the order of colors in the scheme.

`blind` 'character' string. Type of color blindness to simulate: specify `"deutan"` for green-blind vision, `"protan"` for red-blind vision, `"tritan"` for green-blue-blind vision, or `"monochrome"` for total-color blindness. A partial-color blindness simulation requires that the **dichromat** package is available, see `dichromat` function for additional information. Argument choices may be abbreviated as long as there is no ambiguity.

`gray` 'logical' flag. Whether to subset/reorder the `"bright"`, `"high-contrast"`, `"vibrant"`, and `"muted"` schemes to work well after conversion to gray scale.

`...` Not used

Details

The suggested data type for color schemes and the characteristics of generated palettes are given in the tables below. [**Type**: is the type of data being represented, either qualitative, diverging, or sequential. **Max n**: is the maximum number of colors in a generated palette. And the maximum n value when scheme colors are designed for gray-scale conversion is enclosed in parentheses. A value of infinity indicates that the scheme allows for color interpolations. **N**: is the not-a-number color. **B**: is the background color. **F**: is the foreground color. **Abbreviations**: –, not available]

Table 1. Scheme by Anton Mikhailov (2019); released under an open license.

Type	Scheme	Palette	Max n	N	B	F
Sequential	turbo		∞	–	–	–

Table 2. Schemes by Paul Tol (2018) with permission granted to distribute in Oct 2018.

Type	Scheme	Palette	Max n	N	B	F
Diverging	BuRd		∞		–	–
	PRGn		∞		–	–
	sunset		∞		–	–
Qualitative	bright		7 (3)	–	–	–
	dark		6	–	–	–
	ground cover		14	–	–	–
	high-contrast		5 (5)	–	–	–
	light		9	–	–	–
	muted		9 (5)	–	–	–
	pale		6	–	–	–
	vibrant		7 (4)	–	–	–
Sequential	discrete rainbow		23		–	–
	iridescent		∞		–	–
	smooth rainbow		∞		–	–
	YlOrBr		∞		–	–

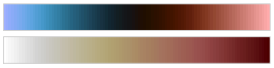
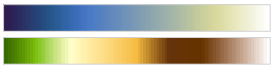
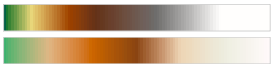
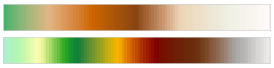
Table 3. Schemes by Thomas Dewez (2004) with permission granted to distribute in Oct 2018.

Type	Scheme	Palette	Max n	N	B	F
Sequential	DEM poster		∞			
	DEM print		∞			
	DEM screen		∞			

Table 4. Scheme by unknown author; discovered on gnuplot-info by Edzer Pebesma.

Type	Scheme	Palette	Max n	N	B	F
Sequential	bpy		∞	–	–	–

Table 5. Schemes collected by Wessel and others (2013) and released under an open license.

Type	Scheme	Palette	Max n	N	B	F
Diverging	polar		∞	–	–	–
	red2green		∞	–	–	–
	roma		∞			
	split		∞	–	–	–
Sequential	abyss		∞			
	acton		∞			
	bamako		∞			
	bathy		∞			
	batlow		∞			
	berlin		∞			
	bilbao		∞			
	broc		∞			
	buda		∞			
	cool		∞	–	–	–
	copper		∞	–	–	–
	cork		∞			
	cubhelix		∞			
	davos		∞			
	dem1		∞			
	dem2		∞			
	dem3		∞			
	dem4		∞			
	devon		∞			
	drywet		∞	–	–	–
	elevation		∞			
	gray		∞	–	–	–
	grayC		∞			
	hawaii		∞			
	hot		∞	–	–	–
	imola		∞			
	inferno		∞	–	–	–
	jet		∞	–	–	–
	lajolla		∞			
	lapaz		∞			
	lisbon		∞			
	magma		∞	–	–	–
	nuuk		∞			
	ocean		∞	–		
	oslo		∞			
	plasma		∞	–	–	–
	seafloor		∞	–	–	–
	seis		∞	–	–	–
	tofino		∞			
	tokyo		∞			
	turku		∞			
	vik		∞			
	viridis		∞	–	–	–

Schemes "pale", "dark", and "ground cover" are intended to be accessed in their entirety and subset using vector element names.

Value

When argument n is specified the function returns an object of class 'inlpal' that inherits behavior from the 'character' class. And when n is unspecified a variant of the GetColors function is returned that has default argument values set equal to the values specified by the user.

The inlpal-class object is comprised of a 'character' vector of n colors in the RGB color system. Colors are specified with a string of the form "#RRGGBB" or "#RRGGBBAA" where RR, GG, BB, and AA are the red, green, blue, and alpha hexadecimal values (00 to FF), respectively. Attributes of the returned object include: "names", the informal names assigned to colors in the palette, where NULL indicates no color names are specified; "NaN", a 'character' string giving the color meant for missing data, in hexadecimal format, where NA indicates no color is specified; and "call", an object of class 'call' giving the unevaluated function call (expression) that can be used to reproduce the color palette. Use the eval function to evaluate the "call" argument. A simple plot method is provided for the 'inlpal' class that shows a palette of colors using a sequence of shaded rectangles, see 'Examples' section for usage.

Note

Sequential color schemes "YlOrBr" and "iridescent" work well for conversion to gray scale.

Author(s)

J.C. Fisher, U.S. Geological Survey, Idaho Water Science Center

References

Dewez, Thomas, 2004, Variations on a DEM palette, accessed October 15, 2018 at http://soliton.vm.bytemark.co.uk/pub/cpt-city/td/index.html

Mikhailov, Anton, 2019, Turbo, an improved rainbow colormap for visualization: Google AI Blog, accessed August 21, 2019 at https://ai.googleblog.com/2019/08/turbo-improved-rainbow-colormap-for.html.

Tol, Paul, 2018, Colour Schemes: SRON Technical Note, doc. no. SRON/EPS/TN/09-002, issue 3.1, 20 p., accessed September 24, 2018 at https://personal.sron.nl/~pault/data/colourschemes.pdf.

Wessel, P., Smith, W.H.F., Scharroo, R., Luis, J.F., and Wobbe, R., 2013, Generic Mapping Tools: Improved version released, AGU, v. 94, no. 45, p. 409–410 doi:10.1002/2013EO450001

See Also

SetHinge function to set the hinge location in a color palette derived from one or two color schemes.

col2rgb function to express palette colors represented in the hexadecimal format as RGB triplets (R, G, B).

Examples

```
pal <- GetColors(n = 10)
print(pal)
plot(pal)

Pal <- GetColors(scheme = "turbo")
formals(Pal)
filled.contour(datasets::volcano, color.palette = Pal,
               plot.axes = FALSE)

# Diverging color schemes (scheme)
op <- par(mfrow = c(6, 1), oma = c(0, 0, 0, 0))
plot(GetColors(  9, scheme = "BuRd"))
plot(GetColors(255, scheme = "BuRd"))
plot(GetColors(  9, scheme = "PRGn"))
plot(GetColors(255, scheme = "PRGn"))
plot(GetColors( 11, scheme = "sunset"))
plot(GetColors(255, scheme = "sunset"))
par(op)

# Qualitative color schemes (scheme)
op <- par(mfrow = c(7, 1), oma = c(0, 0, 0, 0))
plot(GetColors(7, scheme = "bright"))
plot(GetColors(6, scheme = "dark"))
plot(GetColors(5, scheme = "high-contrast"))
plot(GetColors(9, scheme = "light"))
plot(GetColors(9, scheme = "muted"))
plot(GetColors(6, scheme = "pale"))
plot(GetColors(7, scheme = "vibrant"))
par(op)

# Sequential color schemes (scheme)
op <- par(mfrow = c(7, 1), oma = c(0, 0, 0, 0))
plot(GetColors( 23, scheme = "discrete rainbow"))
plot(GetColors( 34, scheme = "smooth rainbow"))
plot(GetColors(255, scheme = "smooth rainbow"))
plot(GetColors(  9, scheme = "YlOrBr"))
plot(GetColors(255, scheme = "YlOrBr"))
plot(GetColors( 23, scheme = "iridescent"))
plot(GetColors(255, scheme = "iridescent"))
par(op)

# Alpha transparency (alpha)
op <- par(mfrow = c(5, 1), oma = c(0, 0, 0, 0))
plot(GetColors(34, alpha = 1.0))
plot(GetColors(34, alpha = 0.8))
plot(GetColors(34, alpha = 0.6))
plot(GetColors(34, alpha = 0.4))
plot(GetColors(34, alpha = 0.2))
par(op)

# Color stops (stops)
```

```
op <- par(mfrow = c(4, 1), oma = c(0, 0, 0, 0))
plot(GetColors(255, stops = c(0.0, 1.0)))
plot(GetColors(255, stops = c(0.0, 0.5)))
plot(GetColors(255, stops = c(0.5, 1.0)))
plot(GetColors(255, stops = c(0.3, 0.9)))
par(op)

# Interpolation bias (bias)
op <- par(mfrow = c(7, 1), oma = c(0, 0, 0, 0))
plot(GetColors(255, bias = 0.4))
plot(GetColors(255, bias = 0.6))
plot(GetColors(255, bias = 0.8))
plot(GetColors(255, bias = 1.0))
plot(GetColors(255, bias = 1.2))
plot(GetColors(255, bias = 1.4))
plot(GetColors(255, bias = 1.6))
par(op)

# Reverse colors (reverse)
op <- par(mfrow = c(2, 1), oma = c(0, 0, 0, 0),
          cex = 0.7)
plot(GetColors(10, reverse = FALSE))
plot(GetColors(10, reverse = TRUE))
par(op)

# Color blindness (blind)
op <- par(mfrow = c(5, 1), oma = c(0, 0, 0, 0))
plot(GetColors(34, blind = NULL))
plot(GetColors(34, blind = "deutan"))
plot(GetColors(34, blind = "protan"))
plot(GetColors(34, blind = "tritan"))
plot(GetColors(34, blind = "monochrome"))
par(op)

# Gray-scale preparation (gray)
op <- par(mfrow = c(8, 1), oma = c(0, 0, 0, 0))
plot(GetColors(3, "bright", gray = TRUE))
plot(GetColors(3, "bright", gray = TRUE,
               blind = "monochrome"))
plot(GetColors(5, "high-contrast", gray = TRUE))
plot(GetColors(5, "high-contrast", gray = TRUE,
               blind = "monochrome"))
plot(GetColors(4, "vibrant", gray = TRUE))
plot(GetColors(4, "vibrant", gray = TRUE,
               blind = "monochrome"))
plot(GetColors(5, "muted", gray = TRUE))
plot(GetColors(5, "muted", gray = TRUE,
               blind = "monochrome"))
par(op)
```

GetDaysInMonth *Get Number of Days in a Year and Month*

Description

Calculate the number of days in a year and month.

Usage

```
GetDaysInMonth(x)
```

Arguments

x	'integer' vector. Year and month, with a required date format of YYYYMM.

Value

A 'integer' vector indicating the number of days for each year and month value in x.

Author(s)

J.C. Fisher, U.S. Geological Survey, Idaho Water Science Center

Examples

```
GetDaysInMonth(c("199802", "199804", "200412"))
```

GetInsetLocation *Get Location for Inset in Plot*

Description

Calculate *x* and *y* co-ordinates that can be used to position an inset in a plot frame at a specified keyword location.

Usage

```
GetInsetLocation(dx, dy, loc = "bottomright", inset = 0, pad = 0, padin = 0)
```

Arguments

dx, dy	'numeric' number. Width and height of the inset, respectively.
loc	'character' string. Single keyword used to specify the position of the inset in the main plot region: "bottomright", "bottom", "bottomleft", "left", "topleft", "top", "topright", "right", or "center" to denote inset location.
inset	'numeric' vector of length 1 or 2, value is recycled as necessary. Inset distance from the margins as a fraction of the main plot region.
pad	'numeric' vector of length 1 or 2, value is recycled as necessary. Padding distance from the margins in user coordinate units.
padin	'numeric' vector of length 1 or 2, value is recycled as necessary. Padding distance from the margins in inches.

Value

A 'numeric' vector of length 2 giving the user coordinates for the bottom-left corner of the inset.

Author(s)

J.C. Fisher, U.S. Geological Survey, Idaho Water Science Center

Examples

```
plot(NA, NA, xlim = c(0, 100), ylim = c(0, 1),
     xlab = "x", ylab = "y", xaxs = "i", yaxs = "i")
dx <- 20; dy <- 0.2
xy <- GetInsetLocation(dx, dy, loc = "bottomleft")
rect(xy[1], xy[2], xy[1] + dx, xy[2] + dy, border = "red")
points(xy[1], xy[2], pch = 16, xpd = TRUE)
print(xy)

xy <- GetInsetLocation(dx, dy, loc = "bottomleft", inset = 0.05)
rect(xy[1], xy[2], xy[1] + dx, xy[2] + dy, border = "pink")
points(xy[1], xy[2], pch = 16)
print(xy)

xy <- GetInsetLocation(dx, dy, loc = "topright", padin = 0.5)
rect(xy[1], xy[2], xy[1] + dx, xy[2] + dy, border = "blue")

xy <- GetInsetLocation(dx, dy, loc = "left", pad = c(5, 0))
rect(xy[1], xy[2], xy[1] + dx, xy[2] + dy, border = "green")

xy <- GetInsetLocation(dx, dy, loc = "center")
rect(xy[1], xy[2], xy[1] + dx, xy[2] + dy, border = "brown")
```

GetRegionOfInterest *Get Region of Interest*

Description

Create a spatial polygon describing the convex hull of a set of spatial points.

Usage

```
GetRegionOfInterest(x, y = NULL, alpha = NULL, width = NULL, ...)
```

Arguments

x, y	Coordinate vectors of a set of points. Alternatively, a single argument x can be provided. Functions xy.coords and coordinates are used to extract point coordinates.
alpha	'numeric' number. Value of α, used to implement a generalization of the convex hull (Edelsbrunner and others, 1983). As α decreases, the shape shrinks. Requires that the **alphahull** and **maptools** packages are available. Note that the alphahull package is released under a restrictive non-free software license.
width	'numeric' number. Buffer distance from geometry of convex hull.
...	Additional arguments to be passed to the gBuffer function.

Value

An object of class 'SpatialPolygons'.

Author(s)

J.C. Fisher, U.S. Geological Survey, Idaho Water Science Center

References

Edelsbrunner, H., Kirkpatrick, D.G. and Seidel, R., 1983, On the shape of a set of points in the plane: IEEE Transactions on Information Theory, v. 29, no. 4, p. 551–559.

See Also

Functions chull and ashape are used to calculate the convex hull and generalized convex hull, respectively.

Function checkPolygonsHoles is used to identify polygon holes.

Examples

```
set.seed(123)

n <- 50
x <- list("x" = stats::runif(n), "y" = stats::runif(n))
sp::plot(GetRegionOfInterest(x, width = 0.05), border = "blue", lty = 2)
sp::plot(GetRegionOfInterest(x), border = "red", add = TRUE)
sp::plot(GetRegionOfInterest(x, width = -0.05), lty = 2, add = TRUE)
points(x, pch = 3)

## Not run:
n <- 300
theta <- stats::runif(n, 0, 2 * pi)
r <- sqrt(stats::runif(n, 0.25^2, 0.50^2))
x <- sp::SpatialPoints(cbind(0.5 + r * cos(theta), 0.5 + r * sin(theta)),
                       proj4string = sp::CRS("+init=epsg:32610"))
sp::plot(GetRegionOfInterest(x, alpha = 0.1, width = 0.05),
         col = "green")
sp::plot(GetRegionOfInterest(x, alpha = 0.1),
         col = "yellow", add = TRUE)
sp::plot(x, add = TRUE)

## End(Not run)
```

`Grid2Polygons` *Convert Spatial Grids to Polygons*

Description

Convert gridded spatial data to spatial polygons. Image files created with spatial polygons are reduced in size, can easily be transformed from one coordinate reference system to another, and result in much "cleaner" images when plotted.

Usage

```
Grid2Polygons(
  grd,
  zcol = 1,
  level = FALSE,
  at = NULL,
  cuts = 20,
  pretty = FALSE,
  xlim = NULL,
  ylim = NULL,
  zlim = NULL,
  ply = NULL,
  check_validity = TRUE
)
```

Arguments

grd	'SpatialGridDataFrame', 'SpatialPixelsDataFrame', or 'Raster*'. Spatial grid
zcol	'character' string or 'integer' count. Layer to extract from a multi-layer spatial grid.
level	'logical' flag. If true, a set of levels is used to partition the range of attribute values, its default is false.
at	'numeric' vector. Breakpoints along the range of attribute values.
cuts	'integer' count. Number of levels the range of attribute values would be divided into.
pretty	'logical' flag. Whether to use pretty cut locations.
xlim	'numeric' vector of length 2. Left and right limits of the spatial grid, data outside these limits is excluded.
ylim	'numeric' vector of length 2. Lower and upper limits of the spatial grid, data outside these limits is excluded.
zlim	'numeric' vector of length 2. Minimum and maximum limits of the attribute variable, data outside these limits is excluded.
ply	'SpatialPolygons', or 'SpatialGridDataFrame'. Cropping polygon
check_validity	'logical' flag. If true (default), check the validity of polygons. If any of the polygons are invalid, try making them valid by zero-width buffering.

Value

An object of class 'SpatialPolygonsDataFrame'. The objects `data` slot is a data frame, number of rows equal to the number of `Polygons` objects and a single column containing attribute values. If `level` is true, attribute values are set equal to the midpoint between breakpoints. The status of the polygon as a hole or an island is taken from the ring direction, with clockwise meaning island, and counter-clockwise meaning hole.

Note

The traditional R graphics model does not draw polygon holes correctly, holes overpaint their containing 'Polygon' object using a user defined background color (white by default). Polygon holes are now rendered correctly using the `plot` method for spatial polygons (`SpatialPolygons-class`), see `polypath` for more details. The Trellis graphics model appears to rely on the traditional method so use caution when plotting with `spplot`.

Author(s)

J.C. Fisher, U.S. Geological Survey, Idaho Water Science Center

See Also

As an alternative, consider using the `rasterToPolygons` function in the **raster** package, setting `dissolve = TRUE`.

Examples

```
# Example 1
z <- c(1.1,  1.5,  4.2,  4.1,  4.3,  4.7,
       1.2,  1.4,  4.8,  4.8,   NA,  4.1,
       1.7,  4.2,  1.4,  4.8,  4.0,  4.4,
       1.1,  1.3,  1.2,  4.8,  1.6,   NA,
       3.3,  2.9,   NA,  4.1,  1.0,  4.0)
m <- 5
n <- 6
x <- rep(0:n, m + 1)
y <- rep(0:m, each = n + 1)
xc <- c(rep(seq(0.5, n - 0.5, by = 1), m))
yc <- rep(rev(seq(0.5, m - 0.5, by = 1)), each = n)
grd <- data.frame(z = z, xc = xc, yc = yc)
sp::coordinates(grd) <- ~ xc + yc
sp::gridded(grd) <- TRUE
grd <- as(grd, "SpatialGridDataFrame")
image(grd, col = gray.colors(30), axes = TRUE)
grid(col = "black", lty = 1)
points(x = x, y = y, pch = 16)
text(cbind(xc, yc), labels = z)
text(cbind(x = x + 0.1, y = rev(y + 0.1)),
     labels = 1:((m + 1) * (n + 1)), cex = 0.6)
at <- 1:ceiling(max(z, na.rm = TRUE))
plys <- Grid2Polygons(grd, level = TRUE, at = at)
cols <- GetColors(length(plys), scheme = "bright", alpha = 0.3)
sp::plot(plys, add = TRUE, col = cols)
zz <- plys[[1]]
legend("top", legend = zz, fill = cols, bty = "n", xpd = TRUE,
       inset = c(0, -0.1), ncol = length(plys))

v1 <- rbind(c( 1.2, 0.5), c(5.8, 1.7), c( 2.5, 5.1), c( 1.2, 0.5))
v2 <- rbind(c( 2.5, 2.5), c(3.4, 1.8), c( 3.7, 3.1), c( 2.5, 2.5))
v3 <- rbind(c(-0.3, 3.3), c(1.7, 5.1), c(-1.0, 7.0), c(-0.3, 3.3))
p1 <- sp::Polygon(v1, hole = FALSE)
p2 <- sp::Polygon(v2, hole = TRUE)
p3 <- sp::Polygon(v3, hole = FALSE)
p <- sp::SpatialPolygons(list(sp::Polygons(list(p1, p2, p3), 1)))
plys <- Grid2Polygons(grd, level = TRUE, at = at, ply = p)
cols <- GetColors(length(zz), scheme = "bright", alpha = 0.6)
cols <- cols[zz %in% plys[[1]]]
sp::plot(plys, col = cols, add = TRUE)
text(cbind(xc, yc), labels = z)

# Example 2
data(meuse.grid, package = "sp")
sp::coordinates(meuse.grid) <- ~ x + y
sp::gridded(meuse.grid) <- TRUE
meuse.grid <- as(meuse.grid, "SpatialGridDataFrame")
meuse.plys <- Grid2Polygons(meuse.grid, "dist", level = FALSE)
op <- par(mfrow = c(1, 2), oma = rep(0, 4), mar = rep(0, 4))
sp::plot(meuse.plys, col = heat.colors(length(meuse.plys)))
```

```
title("level = FALSE", line = -7)
meuse.plys.lev <- Grid2Polygons(meuse.grid, "dist", level = TRUE)
sp::plot(meuse.plys.lev, col = heat.colors(length(meuse.plys.lev)))
title("level = TRUE", line = -7)
par(op)

# Example 3
m <- datasets::volcano
m <- m[nrow(m):1, ncol(m):1]
x <- seq(from = 2667405, length.out = ncol(m), by = 10)
y <- seq(from = 6478705, length.out = nrow(m), by = 10)
r <- raster::raster(m, xmn = min(x), xmx = max(x), ymn = min(y),
                    ymx = max(y), crs = "+init=epsg:27200")
plys <- Grid2Polygons(r, level = TRUE)
cols <- GetColors(length(plys), scheme = "DEM screen")
sp::plot(plys, col = cols, border = "#515151")
```

PlotCrossSection *Plot Cross Section*

Description

Draw a cross-section view of raster data. A key showing how the colors map to raster values is shown below the map. The width and height of the graphics region will be automagically determined in some cases.

Usage

```
PlotCrossSection(
  transect,
  rs,
  geo.lays = names(rs),
  val.lays = NULL,
  wt.lay = NULL,
  asp = 1,
  ylim = NULL,
  max.dev.dim = c(43, 56),
  n = NULL,
  breaks = NULL,
  pal = NULL,
  col = NULL,
  ylab = NULL,
  unit = NULL,
  id = c("A", "A'"),
  labels = NULL,
  explanation = NULL,
  features = NULL,
```

```
  max.feature.dist = Inf,
  draw.key = TRUE,
  draw.sep = TRUE,
  is.categorical = FALSE,
  contour.lines = NULL,
  bg.col = NULL,
  wt.col = "#FFFFFFD8",
  bend.label = "BEND",
  scale.loc = NULL,
  file = NULL
)
```

Arguments

transect	'SpatialLines'. Piecewise linear transect line.
rs	'RasterStack' or 'RasterBrick'. Collection of raster layers with the same extent and resolution.
geo.lays	'character' vector. Names in rs that specify the geometry-raster layers; these must be given in decreasing order, that is, from the upper most (such as land surface) to the lowest (such as a bedrock surface).
val.lays	'character' vector. Names in rs that specify the value-raster layers (optional). Values from the first layer are mapped as colors to the area between the first and second geometry layers; the second layer mapped between the second and third geometry layers, and so on.
wt.lay	'character' string. Name in rs that specifies the water-table-raster layer (optional).
asp	'numeric' number. *y/x* aspect ratio for spatial axes. Defaults to 1 (one unit on the *x*-axis equals one unit on the *y*-axis) when r is projected, otherwise, a calculated value based on axes limits is used.
ylim	'numeric' vector of length 2. Minimum and maximum values for the *y*-axis.
max.dev.dim	'numeric' vector of length 2, value is recycled as necessary. Maximum width and height for the graphics device in picas, respectively. Where 1 pica is equal to 1/6 of an inch, 4.2333 of a millimeter, or 12 points. Suggested dimensions for single-column, double-column, and side title figures are c(21,56), c(43,56), and c(56,43), respectively. This argument is only applicable when the file argument is specified.
n	'integer' count. Desired number of intervals to partition the range of raster values (optional).
breaks	'numeric' vector. Break points used to partition the colors representing numeric raster values (optional).
pal	'function'. Color palette to be used to assign colors in the plot.
col	'character' vector. Colors to be used in the plot. This argument requires breaks specification for numeric raster values and overrides any palette function specification. For numeric values there should be one less color than breaks. Categorical data require a color for each category.

ylab	'character' string. Label for the *y* axis.
unit	'character' string. Label for the measurement unit of the *x*- and *y*-axes.
id	'character' vector of length 2. Labels for the end points of the transect line, defaults to *A–A'*.
labels	'list'. Location and values of labels in the color key. This list may include components at and labels.
explanation	'character' string. Label explaining the raster cell value.
features	'SpatialPointsDataFrame'. Point features adjacent to the transect line that are used as reference labels for the upper geometry layer. Labels taken from first column of embedded data table.
max.feature.dist	'numeric' number. Maximum distance from a point feature to the transect line, specified in the units of the rs projection.
draw.key	'logical' flag. Whether a color key should be drawn.
draw.sep	'logical' flag. Whether lines separating geometry layers are drawn.
is.categorical	'logical' flag. If true, cell values in val.lays represent categorical data; otherwise, these data values are assumed continuous.
contour.lines	'list'. If specified, contour lines are drawn. The contours are described using a list of arguments supplied to the contour function. Passed arguments include drawlables, method, and col.
bg.col	'character' string. Color used for the background of the area below the top geometry-raster layer.
wt.col	'character' string. Color used for the water-table line.
bend.label	'character' vector. Labels to place at top of the bend-in-section lines, values are recycled as necessary to the number of bends.
scale.loc	'character' string. Position of the scale bar in the main plot region; see GetInsetLocation function for keyword descriptions.
file	'character' string. Name of the output file. Specifying this argument will start a graphics device driver for producing a PDF or PNG file format—the file extension determines the format type. The width and height of the graphics region will be automagically determined and included with the function's returned values, see "Value" section for details; these device dimensions can be useful when creating similar map layouts in dynamic reports.

Value

A 'list' with the following graphical parameters:

din device dimensions (width,height), in inches.

usr extremes of the coordinates of the plotting region (x1,x2,y1,y2).

heights relative heights on the device (upper,lower) for the map and color-key plots.

Author(s)

J.C. Fisher, U.S. Geological Survey, Idaho Water Science Center

See Also

AddScaleBar, AddColorKey, ExtractAlongTransect

Examples

```
m <- datasets::volcano
m <- m[nrow(m):1, ncol(m):1]
x <- seq(from = 2667405, length.out = ncol(m), by = 10)
y <- seq(from = 6478705, length.out = nrow(m), by = 10)
r1 <- raster::raster(m, xmn = min(x), xmx = max(x), ymn = min(y),
                     ymx = max(y), crs = "+init=epsg:27200")
r2 <- min(r1[]) - r1 / 10
r3 <- r1 - r2
rs <- raster::stack(r1, r2, r3)
names(rs) <- c("r1", "r2", "r3")
xy <- rbind(c(2667508, 6479501),
            c(2667803, 6479214),
            c(2667508, 6478749))
transect <- sp::Lines(list(sp::Line(xy)), ID = "Transect")
transect <- sp::SpatialLines(list(transect),
                             proj4string = raster::crs(rs))
xy <- rbind(c(2667705, 6478897),
            c(2667430, 6479178))
p <- sp::SpatialPoints(xy, proj4string = raster::crs(rs))
d <-  data.frame("label" = c("Peak", "Random"))
features <- sp::SpatialPointsDataFrame(p, d, match.ID = TRUE)
bg.image <- raster::hillShade(raster::terrain(r1, "slope"),
                              raster::terrain(r1, "aspect"))
PlotMap(r1, bg.image = bg.image,
        pal = GetColors(scheme = "DEM screen", alpha = 0.8),
        scale.loc = "top", arrow.loc = "topright",
        contour.lines = list("col" = "#1F1F1FA6"),
        useRaster = TRUE)
lines(transect)
raster::text(as(transect, "SpatialPoints"),
             labels = c("A", "BEND", "A'"),
             halo = TRUE, cex = 0.7, pos = c(3, 4, 1),
             offset = 0.1, font = 4)
points(features, pch = 19)
raster::text(features, labels = features@data$label, halo = TRUE,
             cex = 0.7, pos = 4, offset = 0.5, font = 4)

dev.new()
asp <- 5
unit <- "METERS"
explanation <- "Vertical thickness between layers, in meters."
PlotCrossSection(transect, rs, geo.lays = c("r1", "r2"),
                 val.lays = "r3", ylab = "Elevation", asp = asp,
                 unit = unit, explanation = explanation,
                 features = features, max.feature.dist = 100,
                 bg.col = "#E1E1E1", bend.label = "BEND IN\nSECTION",
                 scale.loc = NULL)
```

```
AddScaleBar(unit = unit, vert.exag = asp, inset = 0.05)

val <- PlotCrossSection(transect, rs, geo.lays = c("r1", "r2"),
                        val.lays = "r3", ylab = "Elevation", asp = 5,
                        unit = "METERS", explanation = explanation,
                        file = "Rplots.png")
print(val)

graphics.off()
file.remove("Rplots.png")
```

PlotGraph *Plot Graph*

Description

Draw a sequence of points, lines, or box-and-whiskers.

Usage

```
PlotGraph(
  x,
  y,
  xlab,
  ylab,
  main = NULL,
  asp = NA,
  xlim = NULL,
  ylim = NULL,
  xn = 5,
  yn = 5,
  ylog = FALSE,
  type = "s",
  lty = 1,
  lwd = 0.7,
  pch = NULL,
  col = NULL,
  bg = NA,
  fill = "none",
  fillcolor = NULL,
  pt.cex = 1,
  xpd = FALSE,
  seq.date.by = NULL,
  scientific = NA,
  conversion.factor = NULL,
  boxwex = 0.8,
  center.date.labels = FALSE,
```

```
  bg.polygon = NULL,
  add.grid = TRUE
)
```

Arguments

x, y	'Date' vector, 'numeric' vector, 'matrix', or 'data.frame'. Data for plotting where the vector length or number of rows should match. If y is missing, then x = x[,1] and y = x[,-1].
xlab	'character' string. Title for *x* axis.
ylab	'character' vector of length 2. Title for the 1st and 2nd-*y* axes. The title for the 2nd-*y* axis is optional and requires conversion.factor be specified.
main	'character' string. Main title for the plot.
asp	'numeric' number. *y/x* aspect ratio for spatial axes. Defaults to 1 (one unit on the *x*-axis equals one unit on the *y*-axis) when r is projected, otherwise, a calculated value based on axes limits is used.
xlim	'numeric' or 'Date' vector of length 2. Minimum and maximum values for the *x*-axis.
ylim	'numeric' vector of length 2. Minimum and maximum values for the *y*-axis.
xn, yn	'integer' count. Desired number of intervals between tick-marks on the *x*- and *y*-axis, respectively.
ylog	'logical' flag. Whether a logarithm scale is used for the *y* axis.
type	'character' string. Plot type, possible types are • "p" for **p**oints, • "l" for **l**ines, • "b" for **b**oth points and lines, • "s" for stair **s**teps (default), • "w" for box-and-**w**hisker, • "i" for **i**nterval-censored data, see "Details" section below, and • "n" for **n**o plotting.
lty	'integer' vector. Line type, see par function for all possible types. Line types are used cyclically.
lwd	'numeric' number. Line width
pch	'integer' count. Point type, see points function for all possible types.
col	'character' vector or 'function'. Point or line color, see par function for all possible ways this can be specified. Colors are used cyclically.
bg	'character' vector. Background colors for the open plot symbols given by pch = 21:25 as in points.
fill	'character' string. Used to create filled area plots. Specify "tozeroy" to fill to zero on the *y*-axis; "tominy" to fill to the minimum *y* value in the plotting region; and "tomaxy" to fill to the maximum. Requires plot type = "l", "b", and "s".

`fillcolor` 'character' vector. Colors for basic filled area plots. Defaults to a half-transparent variant of the line color (`col`).

`pt.cex` 'numeric' number. Expansion factor for the point symbols.

`xpd` 'logical' flag. Whether to prevent point and (or) line symbols from being clipped to the plot region.

`seq.date.by` 'character' string, 'numeric' number, or 'difftime'. The increment of the date sequence, see the `by` argument in the `seq.Date` function for all possible ways this can be specified.

`scientific` 'logical' vector of length 1, 2, or 3, value is recycled as necessary. Whether axes labels should be encoded in nice scientific format. Vector elements correspond to the *x*-axis, `y`-axis, and second *y*-axis labels. Values are recycled as necessary. Missing values correspond to the current default penalty (see `options("scipen")`) to be applied when deciding to print numeric values in fixed or scientific notation.

`conversion.factor` 'numeric' number. Conversion factor for the 2nd-*y* axis.

`boxwex` 'numeric' number. Scale factor to be applied to all boxes, only applicable for box-and-whisker plots.

`center.date.labels` 'logical' flag. If true, date labels are horizontally centered between *x*-axis tick-marks.

`bg.polygon` 'list'. If specified, a background polygon is drawn. The polygon is described using a list of arguments supplied to the `polygon` function. Passed arguments include `"x"` and `"col"`.

`add.grid` 'logical' flag. Whether to draw a rectangular grid.

Details

Interval censored data (`type = "i"`) requires `y` be matrix of 2 columns. The first column contains the starting values, the second the ending values. Observations are represented using `(y0,Inf)` for right-censored value, `(y0,y0)` for exact value, and `(-Inf,y1)` for left-censored value, and `(y0,y1)` for an interval censored value. Where infinity is represented as `Inf` or `NA`, and `y` is a numeric value.

Value

Invisible `NULL`

Author(s)

J.C. Fisher, U.S. Geological Survey, Idaho Water Science Center

See Also

`AddIntervals`

Examples

```
n <- 50L
x <- as.Date("2008-07-12") + 1:n
y <- sample.int(n, replace = TRUE)
PlotGraph(x, y, ylab = paste("Random number in", c("meters", "feet")),
          main = "Main Title", type = "p", pch = 16,
          scientific = FALSE, conversion.factor = 3.28)

y <- data.frame(lapply(1:3, function(i) sample(n, replace = TRUE)))
PlotGraph(x, y, ylab = "Random number", pch = 1,
          seq.date.by = "days", scientific = TRUE)

y <- sapply(1:3, function(i) {
  sample((1:100) + i * 100, n, replace = TRUE)
})
m <- cbind(as.numeric(x), y)
col <- GetColors(3, scheme = "bright")
PlotGraph(m, xlab = "Number", ylab = "Random number", type = "b",
          pch = 15:17, col = col, pt.cex = 0.9)
legend("topright", LETTERS[1:3], inset = 0.02, col = col, lty = 1,
       pch = 15:17, pt.cex = 0.9, cex = 0.7, bg = "white")

d <- data.frame(x = as.Date("2008-07-12") + 1:8 * 1000,
                y0 = c(NA, NA, 1, 3, 1, 4, 2, pi),
                y1 = c(1, 2, NA, NA, 4, 3, 2, pi))
PlotGraph(d, type = "i", ylim = c(0, 5), xpd = TRUE)
```

PlotMap *Plot Map*

Description

Draw a map of raster data and geographical features. A key showing how the colors map to raster values is shown below the map. The width and height of the graphics region will be automagically determined in some cases.

Usage

```
PlotMap(
  r,
  layer = 1,
  att = NULL,
  n = NULL,
  breaks = NULL,
  xlim = NULL,
  ylim = NULL,
  zlim = NULL,
  asp = NULL,
```

```
  extend.xy = FALSE,
  extend.z = FALSE,
  reg.axs = TRUE,
  dms.tick = FALSE,
  bg.lines = FALSE,
  bg.image = NULL,
  bg.image.alpha = 1,
  pal = NULL,
  col = NULL,
  max.dev.dim = c(43, 56),
  labels = NULL,
  scale.loc = NULL,
  arrow.loc = NULL,
  explanation = NULL,
  credit = NULL,
  shade = NULL,
  contour.lines = NULL,
  rivers = NULL,
  lakes = NULL,
  roads = NULL,
  draw.key = NULL,
  draw.raster = TRUE,
  file = NULL,
  close.file = TRUE,
  useRaster,
  simplify
)
```

Arguments

r	'Raster*', 'Spatial*', or 'CRS'. Object that can be converted to a raster layer, or coordinate reference system (CRS).
layer	'integer' count. Layer to extract from if r is of class 'RasterStack/Brick' or 'SpatialGridDataFrame'.
att	'integer' count or 'character' string. Levels attribute to use in the Raster Attribute Table (RAT); requires r values of class factor.
n	'integer' count. Desired number of intervals to partition the range of raster values (or zlim if specified) (optional).
breaks	'numeric' vector. Break points used to partition the colors representing numeric raster values (optional).
xlim	'numeric' vector of length 2. Minimum and maximum values for the *x*-axis.
ylim	'numeric' vector of length 2. Minimum and maximum values for the *y*-axis.
zlim	'numeric' vector of length 2. Minimum and maximum raster values for which colors should be plotted.
asp	'numeric' number. *y/x* aspect ratio for spatial axes. Defaults to 1 (one unit on the *x*-axis equals one unit on the *y*-axis) when r is projected, otherwise, a calculated value based on axes limits is used.

extend.xy	'logical' flag. If true, the spatial limits will be extended to the next tick mark on the axes beyond the grid extent.
extend.z	'logical' flag. If true, the raster value limits will be extended to the next tick mark on the color key beyond the measured range. Not used if the zlim argument is specified.
reg.axs	'logical' flag. If true, the spatial data range is extended.
dms.tick	'logical' flag. If true and r is projected, the axes tickmarks are specified in degrees, minutes, and decimal seconds (DMS).
bg.lines	'logical' flag. If true, grids or graticules are drawn in back of the raster layer using white lines and a grey background.
bg.image	'RasterLayer'. An image to be drawn in back of the main raster layer r, image colors are derived from a vector of gray levels. Raster values typically represent hill shading based on the slope and aspect of land-surface elevations, see hillShade function.
bg.image.alpha	'numeric' number. Opacity of the background image from 0 to 1.
pal	'function'. Color palette to be used to assign colors in the plot.
col	'character' vector. Colors to be used in the plot. This argument requires breaks specification for numeric values of r and overrides any palette function specification. For numeric values there should be one less color than breaks. Factors require a color for each level.
max.dev.dim	'numeric' vector of length 2, value is recycled as necessary. Maximum width and height for the graphics device in picas, respectively. Where 1 pica is equal to 1/6 of an inch, 4.2333 of a millimeter, or 12 points. Suggested dimensions for single-column, double-column, and side title figures are c(21,56), c(43,56), and c(56,43), respectively. This argument is only applicable when the file argument is specified.
labels	'list'. Location and values of labels in the color key. This list may include components at and labels.
scale.loc	'character' string. Position of the scale bar in the main plot region; see GetInsetLocation function for keyword descriptions.
arrow.loc	'character' string. Position of the north arrow in the main plot region; see GetInsetLocation function for keyword descriptions.
explanation	'character' string. Label explaining the raster cell value.
credit	'character' string. Label crediting the base map.
shade	'list'. If specified, a semi-transparent shade layer is drawn on top of the raster layer. This layer is described using a list of arguments supplied to the hillShade function. Passed arguments include angle and direction. Additional arguments also may be passed that control the vertical aspect ratio (z.factor) and color opacity (alpha).
contour.lines	'list'. If specified, contour lines are drawn. The contours are described using a list of arguments supplied to the contour function. Passed arguments include drawlables, method, and col.

rivers	'list'. If specified, lines are drawn. The lines are described using a list of arguments supplied to the plot method for class 'SpatialLines'. Passed arguments include x, col, and lwd.
lakes	'list'. If specified, polygons are drawn. The polygons are described using a list of arguments supplied to the plot method for class 'SpatialPolygons'. Passed arguments include x, col, border, and lwd. Bitmap images require a regular grid.
roads	'list'. If specified, lines are drawn. The lines are described using a list of arguments supplied to the plot method for class 'SpatialLines'. Passed arguments include x, col, and lwd.
draw.key	'logical' flag. Whether a color key should be drawn.
draw.raster	'logical' flag. Whether the raster image should be drawn.
file	'character' string. Name of the output file. Specifying this argument will start a graphics device driver for producing a PDF or PNG file format—the file extension determines the format type. The width and height of the graphics region will be automagically determined and included with the function's returned values, see "Value" section for details; these device dimensions can be useful when creating similar map layouts in dynamic reports.
close.file	'logical' flag. Whether the graphics device driver should be shut down after the function exits. Unused if file = NULL
useRaster	'logical' flag. If true, a bitmap raster is used to plot r instead of using individual polygons for each raster cell. If UseRaster is not specified, raster images are used when the getOption("preferRaster") is true. Unused if simplify = TRUE.
simplify	'numeric' number. Specifying this argument will convert the raster r to spatial polygons prior to plotting, see Grid2Polygons function for details. If simplify > 0 the geometry of the spatial polygons is generalized using the Douglas-Peucker algorithm (Douglas and Peucker, 1961); and simplify is the numerical tolerance value to be used by the algorithm. See gSimplify function for additional information.

Value

A 'list' with the following graphical parameters:

din device dimensions c(width,height), in inches.

usr extremes of the coordinates of the plotting region c(x1,x2,y1,y2).

heights relative heights on the device c(upper,lower) for the map and color-key plots.

Author(s)

J.C. Fisher, U.S. Geological Survey, Idaho Water Science Center

References

Douglas, D., and Peucker, T., 1961, Algorithms for the reduction of the number of points required to represent a digitized line or its caricature: The Canadian Cartographer, v. 10, no. 2, p. 112–122.

See Also

AddColorKey

Examples

```
r <- raster::raster(nrow = 10, ncol = 10, crs = NA)
r[] <- 1L
r[51:100] <- 2L
r[3:6, 1:5] <- 8L
r <- raster::ratify(r)
rat <- cbind(raster::levels(r)[[1]],
             land.cover = c("Pine", "Oak", "Meadow"))
levels(r) <- rat
PlotMap(r)

data(meuse, meuse.grid, package = "sp")
sp::coordinates(meuse.grid) <- ~x+y
sp::proj4string(meuse.grid) <- sp::CRS("+init=epsg:28992")
sp::gridded(meuse.grid) <- TRUE
meuse.grid <- raster::raster(meuse.grid, layer = "soil")
model <- gstat::gstat(id = "zinc", formula = zinc~1,
                      locations = ~x+y, data = meuse)
r <- raster::interpolate(meuse.grid, model)
r <- raster::mask(r, meuse.grid)
Pal <- function(n) GetColors(n, stops=c(0.3, 0.9))
breaks <- seq(0, 2000, by = 200)
credit <- paste("Data collected in a flood plain of the river Meuse,",
                "near the village of Stein (Netherlands),",
                "\nand iterpolated on a grid with 40m by 40m spacing",
                "using inverse distance weighting.")
PlotMap(r, breaks = breaks, pal = Pal, dms.tick = TRUE,
        bg.lines = TRUE, contour.lines = list("col" = "#1F1F1F"),
        credit = credit, draw.key = FALSE, simplify = 0)
AddScaleBar(unit = c("KILOMETER", "MILES"),
            conv.fact = c(0.001, 0.000621371),
            loc = "bottomright", inset = c(0.1, 0.05))
AddGradientLegend(breaks, Pal, at = breaks,
                  title = "Topsoil zinc\nconcentration\n(ppm)",
                  loc = "topleft", inset = c(0.05, 0.1),
                  strip.dim = c(2, 20))

m <- datasets::volcano
m <- m[nrow(m):1, ncol(m):1]
x <- seq(from = 2667405, length.out = ncol(m), by = 10)
y <- seq(from = 6478705, length.out = nrow(m), by = 10)
r <- raster::raster(m, xmn = min(x), xmx = max(x), ymn = min(y),
                    ymx = max(y), crs = "+init=epsg:27200")
bg.image <- raster::hillShade(raster::terrain(r, "slope"),
                              raster::terrain(r, "aspect"))
credit <- paste("Digitized from a topographic map by Ross Ihaka",
                "on a grid with 10-meter by 10-meter spacing.")
explanation <- "Elevation on Auckland's Maunga Whau volcano, in meters."
```

```
PlotMap(r, extend.z = TRUE, bg.image = bg.image,
        pal = GetColors(scheme = "DEM screen", alpha = 0.8),
        scale.loc = "bottomright", arrow.loc = "topright",
        explanation = explanation, credit = credit,
        contour.lines = list("col" = "#1F1F1FA6"), "useRaster" = TRUE)

out <- PlotMap(r, file = "Rplots1.pdf")
print(out)

pdf(file = "Rplots2.pdf", width = out$din[1], height = out$din[2])
PlotMap(r)
raster::contour(r, col = "white", add = TRUE)
dev.off()

file.remove(c("Rplots1.pdf", "Rplots2.pdf"))
graphics.off()
```

`POSIXct2Character` *Convert from POSIXct to Character*

Description

Convert objects from 'POSIXct' class to 'character' class.

Usage

```
POSIXct2Character(x, fmt = "%Y-%m-%d %H:%M:%OS3")
```

Arguments

`x`	'POSIXct' vector. Calendar date and time
`fmt`	'character' string. Conversion specification format

Value

A 'character' vector representing time.

Note

R incorrectly formats objects of calss 'POSIXct' with fractional seconds. For example, a 'POSIXct' time with fractional part `.3` seconds (stored as `0.29999`) is printed as `.2` when represented with one decimal digit. Note that the fractional part on outputs is not rounded. Decimal precision is down to milliseconds on Windows, and down to (almost) microseconds on the other operating systems.

Author(s)

J.C. Fisher, U.S. Geological Survey, Idaho Water Science Center

Examples

```
txt <- c("11/10/2011 07:49:36.3",
         "04/01/2013 17:22:08.123",
         "01/06/2013 01:02:16.123",
         "12/14/2038 15:42:04.123456")
date.time <- as.POSIXct(txt, format = "%m/%d/%Y %H:%M:%OS")

options("digits.secs" = 3)
format(date.time, fmt = "%d/%m/%Y %H:%M:%OS")
format(date.time, fmt = "%d/%m/%Y %H:%M:%OS3")

POSIXct2Character(date.time, fmt = "%d/%m/%Y %H:%M:%OS3")
POSIXct2Character(date.time, fmt = "%d/%m/%Y %H:%M:%OS4")
POSIXct2Character(date.time, fmt = "%d/%m/%Y %H:%M:%OS2")

POSIXct2Character(date.time, fmt = "%H:%M:%OS3 %Y-%m-%d")
```

PrintFigure *Print as LaTeX Figure*

Description

Print the LaTeX code associated with the supplied figure. A figure can be composed of several subfigures and passed to the function as R plotting commands. The applied output format attempts to adhere to the design recommendations for figures in United States Geological Survey (USGS) publications.

Usage

```
PrintFigure(
  fig,
  nr = 1,
  nc = 1,
  label = "",
  title = "",
  title_lof = title,
  headings = "",
  pos = ""
)
```

Arguments

fig	'character' vector. Figure plotting commands written in R. The length of the vector is either equal to the number of subfigures, or 1 when a single plot is desired. An element in the vector contains the commands for creating a single plot.

`nr, nc` 'integer' count. Maximum number of rows and columns in the subfigure layout on a page in the output document.

`label` 'character' string. LaTeX label anchor. Subfigures are labeled using a concatenation of the `label` argument and an index number. For example, specifying `label = "id"` for a figure composed of 3 subfigures results in: labels `"id-1"`, `"id-2"`, and `"id-3"`.

`title` 'character' string. Figure caption

`title_lof` 'character' string. Figure caption to be listed at the beginning of the paper in a "List of Figures".

`headings` 'character' vector. Subfigure captions, values are recycled as necessary to match the vector length of the `fig` argument. To exclude a subfigure caption specify its vector element as `NA`.

`pos` 'character' string. Placement specifiers to be used in `\begin{figure}[pos]`. The specifiers can consist of the following characters in any order:

- `"h"` place the float about at the same point it occurs in the source text;
- `"t"` position at the top of the page;
- `"b"` position at the bottom of the page;
- `"p"` put on a special page for floats only;
- `"!"` override internal parameters LaTeX uses for determining float positions; and
- `"H"` places the float at precisely the location in the source text, requires `\usepackage{float}` in the LaTeX preamble.

Details

Requires `\usepackage{caption}` and `\usepackage{subcaption}` in the LaTeX preamble. The width and height, in inches, to be used in the graphics device (that is, a single plot) are specified in the code-chunk options `fig.width` and `fig.height`, respectively. And always write raw results from R into the output document by also specifying `results = "asis"` in the code-chunk options.

Value

Invisible `NULL`

Author(s)

J.C. Fisher, U.S. Geological Survey, Idaho Water Science Center

Examples

```
## Not run:
cat("\\documentclass{article}",
    "\\usepackage[labelsep=period, labelfont=bf]{caption}",
    "\\usepackage{subcaption}",
    "\\captionsetup[figure]{skip=10pt}",
    "\\captionsetup[subfigure]{skip=0pt, labelfont={bf, it}}",
    "\\renewcommand{\\thesubfigure}{\\Alph{subfigure}}",
```

```
    "\\begin{document}",
    "<<id, echo=FALSE, fig.width=3, fig.height=2, results='asis'>>=",
    "par(mar=c(2.1, 2.1, 1.1, 1.1))",
    "n <- 10",
    "fig <- sprintf('inlmisc::PlotGraph(runif(%s))', 2:n)",
    "headings <- sprintf('Subfigure caption, $n=%s$', 2:n)",
    "PrintFigure(fig, 3, 2, 'id', title='Caption', headings=headings)",
    "@",
    "\\end{document}",
    file = "test-figure.Rnw", sep = "\n")
knitr::knit2pdf("test-figure.Rnw", clean = TRUE)  # requires LaTeX
system("open test-figure.pdf")

unlink(c("test-figure.*", "figure"), recursive = TRUE)

## End(Not run)
```

`PrintPackageHelp` *Print Package Help Documentation*

Description

Print the HTML code associated with the help documentation of one or more R packages.

Usage

```
PrintPackageHelp(
  pkg,
  file = "",
  internal = FALSE,
  toc = FALSE,
  title_to_name = FALSE,
  notrun = TRUE,
  sep = "<hr>",
  links = pkg,
  ...
)
```

Arguments

`pkg`	'character' vector. Package name(s)
`file`	'connection' or 'character' string. Names the file to append output to. Prints to the standard output connection by default.
`internal`	'logical' flag. Whether to print help topics flagged with the keyword "internal".
`toc`	'logical' flag. Whether to format level-2 headers (help-topic titles) using a Markdown syntax. This is required when specifying the table-of-contents (toc) format option in R Markdown, see `rmarkdown::render` function for details.

title_to_name	'logical' flag. Whether to replace the help-topic "title" with its "name".
notrun	'logical' flag. Whether to include ## Not run comments in the Examples section of help documentation.
sep	'character' string. HTML to separate help topics, a horizontal line by default.
links	'character' vector. Names of packages searched (level 0) when creating internal hyperlinks to functions and datasets.
...	Not used

Value

Invisible NULL

Author(s)

J.C. Fisher, U.S. Geological Survey, Idaho Water Science Center

Examples

```
## Not run:
cat("---",
    "title: \"Help Documentation\"",
    "output:",
    "  html_document:",
    "    toc: true",
    "    toc_float: true",
    "---",
    sep = "\n", file = "test-help.Rmd")
PrintPackageHelp("inlmisc", file = "test-help.Rmd", toc = TRUE,
                 title_to_name = TRUE, notrun = FALSE)
rmarkdown::render("test-help.Rmd")
url <- file.path("file:/", getwd(), "test-help.html")
utils::browseURL(url)

file.remove("test-help.Rmd", "test-help.html")

## End(Not run)
```

PrintTable *Print as LaTeX Table*

Description

Print the LaTeX code associated with the supplied data table. The applied output format attempts to adhere to the design recommendations for tables in United States Geological Survey (USGS) publications.

Usage

```
PrintTable(
  d,
  colheadings = NULL,
  align = NULL,
  digits = NULL,
  label = NULL,
  title = NULL,
  headnotes = NULL,
  footnotes = NULL,
  nrec = nrow(d),
  hline = NULL,
  na = "\\textemdash",
  rm_dup = NULL,
  landscape = FALSE,
  ...
)
```

Arguments

d	'data.frame' or 'matrix'. Data table to print.
colheadings	'character' vector, 'matrix', or 'data.frame'. Column headings. For table objects, rows represent layers of headings (stacked headings). The number of columns (or vector length) must equal the number of columns in argument d. A column heading can span multiple columns by repeating adjacent headings. Use \\\\ to code a line break.
align	'character' vector. Column alignment. Specify "l" to left align, "r" to right align, "c" to center align, and "S" to align on the decimal point.
digits	'integer' vector. Number of digits to display in the corresponding columns.
label	'character' string. LaTeX label anchor. Specifying this argument allows you to easily reference the table within the LaTeX document. For example, when label = "id", use \ref{id} to reference the table within a sentence.
title	'character' string. Table caption
headnotes	'character' string. Label placed below the table caption to provide information pertaining to the caption, to the table as a whole, or to the column headings.
footnotes	'character' string. Label placed at the end of the table to provide explanations of individual entries in the table.
nrec	'integer' vector of length 1 or 2, value is recycled as necessary. Maximum number of records to show on the first page, and every subsequent page, respectively.
hline	'integer' vector. Numbers between 1 and nrow(d) -1 indicating the table rows after which a horizontal line should appear.
na	'character' string. Value to be used for missing values in table entries.
rm_dup	'integer' count. End value of a sequence of column indexes (1:rm_dup). Duplicate values contained in these columns will be set equal to an empty string. Where duplicates in a column are determined from the 'character' vector formed by combining its content with the content from all previous columns in the table.

landscape	'logical' flag. If true, conforming PDF viewers will display the table in landscape orientation. This option requires \usepackage[pdftex]{lscape} in the LaTeX preamble.
...	Additional arguments to be passed to the print.xtable function. The arguments type, hline.after and add.to.row should not be included.

Details

Requires \usepackage{caption}, \usepackage{booktabs}, \usepackage{makecell}, \usepackage{multirow}, and \usepackage{siunitx} in the LaTeX preamble.

Value

Invisible NULL

Author(s)

J.C. Fisher, U.S. Geological Survey, Idaho Water Science Center

Examples

```
d <- datasets::iris[, c(5, 1:4)]
colheadings <- rbind(c("Species \\\\ type", rep("Sepal", 2), rep("Petal", 2)),
                     c("", rep(c("Length", "Width"), 2)))
align <- c("l", "c", "c", "c", "c")
digits <- c(0, 1, 1, 1, 1)
title <- "Measurements of sepal length and width and petal
          length and width for three species of Iris flower."
headnotes <- "\\textbf{Iris Species}: setosa, versicolor, and virginica.
              \\textbf{Abbreviations}: cm, centimeters"
levels(d[[1]]) <- sprintf("%s\\footnotemark[%d]", levels(d[[1]]), 1:3)
footnotes <- sprintf("\\footnotemark[%d] Common name is %s iris.",
                     1:3, c("Wild Flag", "Blue Flag", "Virginia"))
footnotes <- paste(footnotes, collapse = "\\\\")
hline <- utils::tail(which(!duplicated(d[[1]])), -1) - 1L
PrintTable(d, colheadings, align, digits, title = title,
           headnotes = headnotes, footnotes = footnotes,
           hline = hline, nrec = c(41, 42), rm_dup = 1)

## Not run:
sink("test-table.tex")
cat("\\documentclass{article}",
    "\\usepackage{geometry}",
    "\\usepackage[labelsep = period, labelfont = bf]{caption}",
    "\\usepackage{siunitx}",
    "\\sisetup{input-ignore = {,}, input-decimal-markers = {.},",
    "          group-separator = {,}, group-minimum-digits = 4}",
    "\\usepackage{booktabs}",
    "\\usepackage{makecell}",
    "\\usepackage{multirow}",
    "\\usepackage[pdftex]{lscape}",
    "\\makeatletter",
```

```
    "\\setlength{\\@fptop}{0pt}",
    "\\makeatother",
    "\\begin{document}", sep = "\n")
PrintTable(d, colheadings, align, digits, title = title,
           headnotes = headnotes, footnotes = footnotes,
           hline = hline, nrec = c(41, 42), rm_dup = 1)
cat("\\clearpage\n")
PrintTable(datasets::CO2[, c(2, 3, 1, 4, 5)],
           digits = c(0, 0, 0, 0, 1),
           title = "Carbon dioxide uptake in grass plants.",
           nrec = 45, rm_dup = 3)
cat("\\clearpage\n")
digits <- c(1, 0, 1, 0, 2, 3, 2, 0, 0, 0, 0)
PrintTable(datasets::mtcars, digits = digits,
           title = "Motor trend car road tests.",
           landscape = TRUE, include.rownames = TRUE)
cat("\\clearpage\n")
x <- c(1.2, 1.23, 1121.2, 184, NA, pi, 0.4)
d <- data.frame(matrix(rep(x, 4), ncol = 4))
d[, 1] <- prettyNum(d[, 1])
d[, 4] <- formatC(d[, 4], digits = 2, format = "e")
colheadings <- paste("Wide heading", 1:ncol(d))
align <- c("S", "S",
           "S[round-mode = places, round-precision = 2]",
           "S[scientific-notation = true, table-format = 1.2e+1]")
PrintTable(d, colheadings, align)
cat("\\end{document}\n")
sink()
tinytex::pdflatex("test-table.tex")  # requires LaTeX
system("open test-table.pdf")

file.remove("test-table.tex", "test-table.pdf")

## End(Not run)
```

ReadCodeChunks *Read Knitr Code Chunks*

Description

Read **knitr** code chunks into the current session.

Usage

```
ReadCodeChunks(path)
```

Arguments

path	'character' string. Path name of the **knitr** source document ('.Rnw' or '.Rmd'), or R code that has been extracted from a **knitr** source document ('.R').

Details

If the source document is '.Rnw' or '.Rmd' the purl function is used to extract the R code. The R code is read into the current session using a chunk separator of the from ## ----chunk-name (at least four dashes before the chunk name) in the script. Unnamed chunks (that is, chunk-name is missing) will be assigned names like unnamed-chunk-i where i is the chunk number.

Value

A 'list' of length equal to the number of code chunks in path. Each list component is named after its corresponding chunk name (chunk-name). The returned object includes the value of the path argument as an attribute.

Author(s)

J.C. Fisher, U.S. Geological Survey, Idaho Water Science Center

Examples

```
file <- system.file("misc/knitr-markdown.Rmd", package = "inlmisc")
chunks <- ReadCodeChunks(file)
print(chunks)
attr(chunks, "path")

txt <- chunks[c("unnamed-chunk-3", "named-chunk-4")]
eval(parse(text = unlist(txt)))
```

ReadModflowBinary *Read MODFLOW Binary File*

Description

Read binary files output from MODFLOW-based models, the U.S. Geological Survey's three-dimensional finite-difference groundwater model.

Usage

```
ReadModflowBinary(
  path,
  data.type = c("array", "flow"),
  endian = c("little", "big"),
  rm.totim.0 = FALSE
)
```

Arguments

`path`	'character' string. Path to a MODFLOW binary file.
`data.type`	'character' string. Description of how the data were saved. Specify `"array"` for array data (such as hydraulic heads or drawdowns) and `"flow"` for cell-by-cell flow data (budget data).
`endian`	'character' string. Endian-ness (or byte-order) of the binary file.
`rm.totim.0`	'logical' flag. Whether data associated with the stress period at time zero should be removed.

Value

A 'list' of length equal to the number of times data were written to the binary file. List components are as follows:

d matrix of values. The matrix dimensions typically coincide with the horizontal model grid. The exception is for flow data (`data.type = "flow"`) when the cell-by-cell budget file is saved using the ***"COMPACT BUDGET"*** output option; for this case, matrix columns are: cell index (`"icell"`), model-grid layer (`"layer"`), model-grid row (`"row"`), model-grid column (`"column"`), cell-by-cell flow (`"flow"`), and any auxiliary variables saved using the ***"AUXILIARY"*** output option.

kstp time step

kper stress period

desc description of data-type, such as "wells".

layer model-grid layer

delt time-step size

pertim elapsed time in the current stress period.

totim total elapsed time

The layer component (layer) and time components (delt, pertim, totim) are only available for flow data when the cell-by-cell budget file is saved using the ***"COMPACT BUDGET"*** output option.

Author(s)

J.C. Fisher, U.S. Geological Survey, Idaho Water Science Center

See Also

SummariseBudget

Examples

```
path <- system.file("extdata", "ex.hds", package = "inlmisc")
heads <- ReadModflowBinary(path, "array")
image(heads[[1]]$d)
str(heads[[1]])

path <- system.file("extdata", "ex.bud", package = "inlmisc")
```

```
budget <- ReadModflowBinary(path, "flow")
image(budget[[1]]$d)
str(budget[[1]])
str(budget[[11]])
```

RecreateLibrary *Recreate R Library*

Description

Recreate an existing library on a new installation of R. The SavePackageDetails function writes the details of installed packages to a text file. And the RecreateLibrary function reads this file and downloads and installs any 'missing' packages from the Comprehensive R Archive Network (CRAN), CRAN-like repositories, and local package-installation files.

Usage

```
RecreateLibrary(
  file = "R-packages.tsv",
  lib = .libPaths()[1],
  repos = getOption("repos"),
  snapshot = FALSE,
  local = NULL,
  versions = FALSE,
  parallel = TRUE,
  quiet = FALSE
)

SavePackageDetails(file = "R-packages.tsv", lib = .libPaths(), pkg = NULL)
```

Arguments

file	'character' string. Name of the file for reading (or writing) the list of package details. For a file name that does not contain an absolute path, the name is assumed relative to the current working directory (see getwd function). A '.gz' file extension indicates the file is compressed by *gzip*.
lib	'character' vector. Library tree(s) to search through when locating installed packages (see .libPaths function), or the library directory where to install packages.
repos	'character' vector. Base URL(s) of the CRAN-like repositories (includes CRAN) to use when installing packages. For example, the URL of the RStudio sponsored CRAN mirror is "https://cloud.r-project.org/". And the URL of the Geological Survey R Archive Network (GRAN) is "https://owi.usgs.gov/R".

snapshot	'logical' flag, 'Date', or 'character' string. Calendar date for a CRAN snapshot in time, see the Microsoft R Application Network (MRAN) website for details. If true, the snapshot date is read from the first line of the package-details file. A snapshot date can also be specified directly using the required date format, YYYY-MM-DD. This argument masks any CRAN mirror specified in repos.
local	'character' vector. Paths to local repositories. Packages are installed from local files in these directories. Files can contain *binary* builds of packages ('.zip' on Windows and '.tgz' on macOS) or be *source* packages ('.tar.gz').
versions	'logical' flag. If true, installed package versions will be identical to version numbers stored in the package-details file. Only applies to packages from CRAN-like repositories and local package-installation files. Requires that the **devtools** package is available.
parallel	'logical' flag or 'integer' count. Whether to use parallel processes for a parallel install of more than one source package. This argument can also be used to specify the number of cores to employ.
quiet	'logical' flag. Whether to reduce the amount of output.
pkg	'character' vector. Names of package(s) located under lib. Only packages specified in pkg, and the packages that pkg depend on/link to/import/suggest, will be included in the package-details file.

Details

A typical workflow is as follows: Run the SavePackageDetails() command on an older version of R. It will print to a text file a complete list of details for packages located under your current R library tree(s). If the older version of R no longer needed, uninstall it. Then, on a freshly installed version of R with the **inlmisc** package available, run the RecreateLibrary() command. It will download and install the packages listed in the package-details file.

The type of package to download and install from CRAN-like repositories is *binary* on Windows and some macOS builds, and *source* on all others. Package installation from a local '.tar.gz' file is always a source installation. If a package is installed from source, and it contains code that needs compiling, you must have a working development environment. On Windows, install the Rtools collection and have the PATH environment variable set up as required by Rtools. On macOS, install Xcode from the Mac App Store. And on Linux, install a compiler and various development libraries.

Daily snapshots of CRAN are stored on MRAN and available as far back as September 17, 2014. Use the snapshot argument to install older package versions from MRAN. Note that newer versions of R may not be compatible with older versions of packages. To avoid any package installation issues, install the R version that was available from CRAN on the snapshot date.

The package-details file is of the following format:

```
# Date modified: YYYY-MM-DD HH:MM:SS UTC
# R version 9.9.9 (YYYY-MM-DD)
Package Version
name    9.9.9
...     ...
```

Where the first two lines are reserved for the timestamp and R-version number, respectively. And package data are stored in a tabular structure, that is, data-table values are separated by a TAB character. The data format is flexible enough to add additional extraneous metadata and table fields, for example,

```
# Date modified: 2017-08-12 05:14:33 UTC
# R version 3.4.1 (2017-06-30)
# Running under: Windows 10 x64 (build 14393)
# Platform: x86_64-w64-mingw32
Package   Version Priority Depends       Imports
akima     0.6-2   NA       R (>= 2.0.0)  sp
animation 2.5     NA       R (>= 2.14.0) NA
```

Value

The `SavePackageDetails` function returns (invisibly) the MD5 hash of the package-details `file` content. Any changes in the file content will produce a different MD5 hash. Use the `md5sum` function to verify that a file has not been changed. The `RecreateLibrary` function returns (invisibly) `NULL`.

Note

This package-installation method does not offer one-hundred percent reproducibility of existing R libraries. Alternative methods, that offer better reproducibility, are available using the **checkpoint** and **packrat** packages; both of which provide robust tools for dependency management in R.

Author(s)

J.C. Fisher, U.S. Geological Survey, Idaho Water Science Center

Examples

```
# Run on old version of R
SavePackageDetails()

## Not run:
# Run on new version of R, and ensure 'inlmisc' package is available.
repos <- c(CRAN = "https://cloud.r-project.org/",
           GRAN = "https://owi.usgs.gov/R")
if (system.file(package = "inlmisc") == "")
  utils::install.packages("inlmisc", repos = repos["CRAN"],
                          dependencies = TRUE)
inlmisc::RecreateLibrary(repos = repos)

## End(Not run)

# Clean up example
file.remove("R-packages.tsv")
```

`ReplaceInTemplate` *Replace Values in a Template Text*

Description

Replace keys within special markups in a template text with specified values. Pieces of R code can be put into the markups of the template text, and are evaluated during the replacement.

Usage

```
ReplaceInTemplate(text, replacement = list())
```

Arguments

`text`	'character' vector. Template text
`replacement`	'list'. Values to replace in `text`.

Details

Keys are enclosed into markups of the form `$(KEY)` and `@{CODE}`.

Value

A 'character' vector of strings after key replacement.

Author(s)

J.C. Fisher, U.S. Geological Survey, Idaho Water Science Center

References

This code was derived from the sensitivity`::template.replace` function, accessed on Feb 6, 2015.

See Also

`SummariseBudget`

Examples

```
text <- c("Hello $(name)!", "$(a) + $(b) = @{$(a) + $(b)}",
          "pi = @{format(pi, digits = 5)}")
cat(text, sep = "\n")
replacement <- list("name" = "world", "a" = 1, "b" = 2)
cat(ReplaceInTemplate(text, replacement), sep = "\n")
```

`RmSmallCellChunks` *Remove Small Cell Chunks*

Description

Remove small cell chunks from a raster layer, where a cell chunk is defined as a group of connected cells with non-missing values. The cell chunk with the largest surface area is preserved and all others removed.

Usage

```
RmSmallCellChunks(r)
```

Arguments

`r`	'RasterLayer'. Raster grid layer with cell values.

Value

An object of class 'RasterLayer' giving `r` with cell values in the smaller cell chunks set to `NA`.

Author(s)

J.C. Fisher, U.S. Geological Survey, Idaho Water Science Center

Examples

```
set.seed(2)
r <- raster::raster(ncols = 10, nrows = 10)
r[] <- round(runif(raster::ncell(r)) * 0.7)
r <- raster::clump(r)
r <- raster::ratify(r)
PlotMap(r)

r_new <- RmSmallCellChunks(r)
PlotMap(r_new)

graphics.off()
```

SetHinge *Set Hinge Location in Color Palette*

Description

The *hinge* indicates a dramatic color change in a palette that is typically located at the midpoint of the data range. An asymmetrical data range can result in an undesired hinge location, a location that does not necessarily coincide with the break-point in the user's data. This function can be used to specify a hinge location that is appropriate for your data.

Usage

```
SetHinge(
  x,
  hinge,
  scheme = "sunset",
  alpha = NULL,
  reverse = FALSE,
  buffer = 0,
  stops = c(0, 1),
  allow_bias = TRUE
)
```

Arguments

x	'numeric' object that can be passed to the range function with NA's removed. The user's data range.
hinge	'numeric' number. Hinge value (such as, at sea-level) in data units.
scheme	'character' vector of length 1 or 2, value is recycled as necessary. Name of color scheme(s). The color palette is derived from one or two color schemes. The scheme(s) must be suitable for continuous data types and allow for color interpolation. See GetColors function for a list of possible scheme names. Argument choices may be abbreviated as long as there is no ambiguity.
alpha	'numeric' vector of length 1 or 2, value is recycled as necessary. Alpha transparency applied separately on either side of the hinge. Values range from 0 (fully transparent) to 1 (fully opaque). Specify as NULL to exclude the alpha channel value from colors.
reverse	'logical' vector of length 1 or 2, value is recycled as necessary. Whether to reverse the order of colors in the scheme(s). Values applied separately on either side of the hinge.
buffer	'numeric' vector of length 1 or 2, value is recycled as necessary. Color buffer around the hinge measured as a fraction of the color range. Values applied separately on either side of the hinge.
stops	'numeric' vector of length 2. Color stops defined by interval endpoints (between 0 and 1) and used to select a subset of the color palette(s).
allow_bias	'logical' flag. Whether to allow bias in the color spacing.

Value

A 'function' that takes an 'integer' argument (the required number of colors) and returns a 'character' vector of colors.

Author(s)

J.C. Fisher, U.S. Geological Survey, Idaho Water Science Center

Examples

```
Plot <- inlmisc:::plot.inlpal
Pal <- SetHinge(x = c(-3, 7), hinge = 0)
Plot(Pal(n = 19))

x <- datasets::volcano
Pal <- SetHinge(x, hinge = 140, scheme = c("abyss", "dem1"))
filled.contour(x, color.palette = Pal, nlevels = 50,
               plot.axes = FALSE)

# Data range (x)
hinge <- 0; n <- 20
op <- par(mfrow = c(5, 1), oma = c(0, 0, 0, 0))
Plot(SetHinge(c(-10,  0), hinge)(n))
Plot(SetHinge(c( -7,  3), hinge)(n))
Plot(SetHinge(c( -5,  5), hinge)(n))
Plot(SetHinge(c( -3,  7), hinge)(n))
Plot(SetHinge(c(  0, 10), hinge)(n))
par(op)

# Hinge value (hinge)
x <- c(-5, 5); n <- 255
op <- par(mfrow = c(5, 1), oma = c(0, 0, 0, 0))
Plot(SetHinge(x, hinge = -6)(n))
Plot(SetHinge(x, hinge = -2)(n))
Plot(SetHinge(x, hinge =  0)(n))
Plot(SetHinge(x, hinge =  2)(n))
Plot(SetHinge(x, hinge =  6)(n))
par(op)

# Color scheme (scheme)
x <- c(-10, 10); hinge <- -3; n <- 255
op <- par(mfrow = c(3, 1), oma = c(0, 0, 0, 0))
Plot(SetHinge(x, hinge, scheme = "roma")(n))
Plot(SetHinge(x, hinge, scheme = "BuRd")(n))
Plot(SetHinge(x, hinge, scheme = c("ocean", "copper"))(n))
par(op)

# Alpha transparency (alpha)
x <- c(-5, 5); hinge <- 0; scheme <- c("drywet", "hawaii"); n <- 255
op <- par(mfrow = c(4, 1), oma = c(0, 0, 0, 0))
Plot(SetHinge(x, hinge, scheme, alpha = 1.0)(n))
Plot(SetHinge(x, hinge, scheme, alpha = 0.5)(n))
```

```
Plot(SetHinge(x, hinge, scheme, alpha = c(1.0, 0.5))(n))
Plot(SetHinge(x, hinge, scheme, alpha = c(0.5, 1.0))(n))
par(op)

# Reverse colors (reverse)
x <- c(-10, 10); hinge <- -3; n <- 255
op <- par(mfrow = c(6, 1), oma = c(0, 0, 0, 0))
Plot(SetHinge(x, hinge, "roma", reverse = FALSE)(n))
Plot(SetHinge(x, hinge, "roma", reverse = TRUE)(n))
Plot(SetHinge(x, hinge, c("davos", "hawaii"),
              reverse = FALSE)(n))
Plot(SetHinge(x, hinge, c("davos", "hawaii"),
              reverse = TRUE)(n))
Plot(SetHinge(x, hinge, c("davos", "hawaii"),
              reverse = c(TRUE, FALSE))(n))
Plot(SetHinge(x, hinge, c("davos", "hawaii"),
              reverse = c(FALSE, TRUE))(n))
par(op)

# Buffer around hinge (buffer)
x <- c(-5, 5); hinge <- -2; n <- 20
op <- par(mfrow = c(6, 1), oma = c(0, 0, 0, 0))
Plot(SetHinge(x, hinge, buffer = 0.0)(n))
Plot(SetHinge(x, hinge, buffer = 0.2)(n))
Plot(SetHinge(x, hinge, buffer = c(0.4, 0.2))(n))
Plot(SetHinge(x, hinge, c("gray", "plasma"),
              buffer = 0.0)(n))
Plot(SetHinge(x, hinge, c("gray", "plasma"),
              buffer = 0.2)(n))
Plot(SetHinge(x, hinge, c("gray", "plasma"),
              buffer = c(0.2, 0.4))(n))
par(op)

# Color stops (stops)
x <- c(-5, 5); hinge <- 1; n <- 20
op <- par(mfrow = c(6, 1), oma = c(0, 0, 0, 0))
Plot(SetHinge(x, hinge, stops = c(0.0, 1.0))(n))
Plot(SetHinge(x, hinge, stops = c(0.2, 0.8))(n))
Plot(SetHinge(x, hinge, stops = c(0.4, 0.6))(n))
Plot(SetHinge(x, hinge, c("gray", "plasma"),
              stops = c(0.0, 1.0))(n))
Plot(SetHinge(x, hinge, c("gray", "plasma"),
              stops = c(0.2, 0.8))(n))
Plot(SetHinge(x, hinge, c("gray", "plasma"),
              stops = c(0.4, 0.6))(n))
par(op)

# Allow bias (allow_bias)
x <- c(-3, 7); n <- 20
op <- par(mfrow = c(4, 1), oma = c(0, 0, 0, 0))
Plot(SetHinge(x, hinge = 0, allow_bias = TRUE)(n))
Plot(SetHinge(x, hinge = 0, allow_bias = FALSE)(n))
Plot(SetHinge(x, hinge = 4, allow_bias = TRUE)(n))
```

```
Plot(SetHinge(x, hinge = 4, allow_bias = FALSE)(n))
par(op)
```

`SetPolygons` *Overlay Multi-Polygon Objects*

Description

Calculate the intersection or difference between two multi-polygon objects.

Usage

```
SetPolygons(x, y, cmd = c("gIntersection", "gDifference"), buffer.width = NA)
```

Arguments

`x`	'SpatialPolygons*'. Multi-polygon object
`y`	'SpatialPolygons*' or 'Extent'. Multi-polygon object
`cmd`	'character' string. Specifying "gIntersection", the default, cuts out portions of the `x` polygons that overlay the `y` polygons. If "gDifference" is specified, only those portions of the `x` polygons falling outside the `y` polygons are copied to the output polygons.
`buffer.width`	'numeric' number. Expands or contracts the geometry of `y` to include the area within the specified width, see `gBuffer`. Specifying `NA`, the default, indicates no buffer.

Details

This function tests if the resulting geometry is valid. If invalid, an attempt is made to make the geometry valid by zero-width buffering.

Value

An object of class 'SpatialPolygons*'.

Author(s)

J.C. Fisher, U.S. Geological Survey, Idaho Water Science Center

Examples

```
m1a <- rbind(c(17.5, 55.1),
             c(24.7, 55.0),
             c(22.6, 61.1),
             c(16.5, 59.7),
             c(17.5, 55.1))
m1b <- m1a
m1b[, 1] <- m1b[, 1] + 11
p1 <- list(sp::Polygon(m1a, FALSE), sp::Polygon(m1b, FALSE))
p1 <- sp::SpatialPolygons(list(sp::Polygons(p1, 1)))
sp::plot(p1, col = "blue")

m2a <- rbind(c(19.6, 60.0),
             c(35.7, 58.8),
             c(28.2, 64.4),
             c(19.6, 60.0))
m2b <- rbind(c(20.6, 56.2),
             c(30.9, 53.8),
             c(27.3, 51.4),
             c(20.6, 56.2))
p2 <- list(sp::Polygon(m2a, FALSE), sp::Polygon(m2b, FALSE))
p2 <- sp::SpatialPolygons(list(sp::Polygons(p2, 2)))
sp::plot(p2, col = "red", add = TRUE)

p <- SetPolygons(p1, p2, "gIntersection")
sp::plot(p, col = "green", add = TRUE)

p <- SetPolygons(p2, p1, "gDifference")
sp::plot(p, col = "purple", add = TRUE)
```

`SummariseBudget` *Summarize MODFLOW Water Budget*

Description

Summarize MODFLOW volumetric flow rates by boundary condition types. Cell-by-cell flow data is split into subsets, summary statistics computed for each subset, and a summary table returned.

Usage

```
SummariseBudget(budget, desc = NULL, id = NULL)
```

Arguments

`budget`	'character' string or 'list'. Either the path to a MODFLOW cell-by-cell budget file or the object returned from the `ReadModflowBinary` function.
`desc`	'character' vector. Data-type descriptors, such as `c("wells","drains")`. If missing, all data types are summarized.

`id` 'character' string. Name of auxiliary variable, a variable of additional values associated with each cell saved using the ***"AUXILIARY"*** output option.

Details

Subsets are grouped by data type (desc), stress period (kper), time step (kstp), and optional auxiliary variable. Data in the MODFLOW cell-by-cell budget file must be saved using the ***"COMPACT BUDGET"*** output option.

Value

A 'data.table' with the following variables:

desc description of data type, such as "wells".

kper stress period

kstp time step

id auxiliary variable name

delt length of the current time step.

pertim time in the stress period.

totim total elapsed time

count number of cells in each subset.

flow.sum total volumetric flow rate

flow.mean mean volumetric flow rate

flow.median median volumetric flow rate

flow.sd standard deviation of volumetric flow rate.

flow.dir flow direction where "in" and "out" indicate water entering and leaving the groundwater system, respectively.

Author(s)

J.C. Fisher, U.S. Geological Survey, Idaho Water Science Center

See Also

`ReadModflowBinary`

Examples

```
path <- system.file("extdata", "ex.bud", package = "inlmisc")
out <- SummariseBudget(path, desc = "river leakage", id = "iface")
print(out)
```

`ToScientific` *Format for Scientific Notation*

Description

Format numbers in scientific notation $m \times 10^n$.

Usage

```
ToScientific(
  x,
  digits = NULL,
  type = c("latex", "plotmath"),
  na = as.character(NA),
  zero = "0",
  delimiter = "$",
  scipen = NULL,
  big.mark = ",",
  ...
)
```

Arguments

`x`	'numeric' vector. Numbers
`digits`	'integer' count. Desired number of digits after the decimal point.
`type`	'character' string. Specify `"latex"` to return numbers in the LaTeX markup language (default), or `"plotmath"` to return as `plotmath` expressions.
`na`	'character' string. Value to use for missing values (`NA`). By default, no string substitution is made for missing values.
`zero`	'character' string. Value to use for zero values. Specify as `NULL` to prevent string substitution.
`delimiter`	'character' string. Delimiter for LaTeX mathematical mode, inline (`$...$`) by default. Does not apply to missing value strings.
`scipen`	'integer' count. Penalty to be applied when deciding to format numeric values in scientific or fixed notation. Positive values bias towards fixed and negative towards scientific notation: fixed notation will be preferred unless it is more than `scipen` digits wider. Specify `NULL` to format all numbers, with the exception of zero, in scientific notation.
`big.mark`	'character' string. Mark inserted between every big interval before the decimal point. By default, commas are placed every 3 decimal places for numbers larger than 999.
`...`	Not used

Value

When `type = "latex"` returns a 'character' vector of the same length as argument `x`. And when `type = "plotmath"` returns a 'expression' vector of the same length as `x`.

Author(s)

J.C. Fisher, U.S. Geological Survey, Idaho Water Science Center

Examples

```
x <- c(-1e+09, 0, NA, pi * 10^(-5:5))
ToScientific(x, digits = 2, na = "---")

ToScientific(x, digits = 2, scipen = 0)

x <- seq(0, 20000, by = 4000)
ToScientific(x, scipen = 0)

lab <- ToScientific(x, type = "plotmath", scipen = 0)
i <- seq_along(x)
plot(i, type = "n", xaxt = "n", yaxt = "n", ann = FALSE)
axis(1, i, labels = lab)
axis(2, i, labels = lab)
```

---

usgs_article *USGS Article Format*

Description

Format for creating a U.S. Geological Survey (USGS) article.

Usage

```
usgs_article(...)
```

Arguments

`...`	Arguments passed to the `pdf_document` function.

Value

R Markdown output format to pass to `render`

Author(s)

J.C. Fisher, U.S. Geological Survey, Idaho Water Science Center

Examples

```
## Not run:
rmarkdown::draft("myarticle.Rmd",
                 template = "usgs_article",
                 package = "inlmisc")

rmarkdown::render("myarticle/myarticle.Rmd")
system("open myarticle/wrapper.pdf")

unlink("myarticle", recursive = TRUE)

## End(Not run)
```

Index