Package 'Ternary'

February 27, 2020

Version 1.1.4

Title Create Ternary Plots

Description Plots ternary diagrams using the standard graphics functions.
An alternative to 'ggtern', which uses the 'ggplot2' family of plotting functions.

URL https://ms609.github.io/Ternary/,
https://github.com/ms609/Ternary/

BugReports https://github.com/ms609/Ternary/issues/

License GPL (>= 2)

Language en-GB

Depends R (>= 3.2.0)

Imports viridisLite

Suggests knitr, rmarkdown, testthat, vdiffr

LazyData true

VignetteBuilder knitr

Encoding UTF-8

RoxygenNote 6.1.1

NeedsCompilation no

Author Martin R. Smith [aut, cre, cph]
(<https://orcid.org/0000-0001-5660-1727>)

Maintainer Martin R. Smith <martin.smith@durham.ac.uk>

Repository CRAN

Date/Publication 2020-02-27 15:40:03 UTC

R topics documented:

`AddToTernary` *Add elements to ternary plot*

Description

Plot shapes onto a ternary diagram created with `TernaryPlot()`.

Usage

```
AddToTernary(PlottingFunction, coordinates, ...)

TernaryArrows(fromCoordinates, toCoordinates = fromCoordinates, ...)

TernaryLines(coordinates, ...)

TernaryPoints(coordinates, ...)

TernaryPolygon(coordinates, ...)

TernaryText(coordinates, ...)

JoinTheDots(coordinates, ...)
```

Arguments

`PlottingFunction`
: Function to add data to a plot; perhaps one of `points`, `lines` or `text`.

`coordinates`
: A list, matrix, data.frame or vector in which each element (or row) specifies the three coordinates of a point in ternary space.

`...`
: Additional parameters to pass to `PlottingFunction()`. If using `TernaryText()`w, this will likely include the parameter `labels`, to specify the text to plot.

`fromCoordinates, toCoordinates`
: For `TernaryArrows()`, coordinates at which arrows should begin and end; *cf.* `x0`, `y0`, `x1` and `y1` in arrows. Recycled as necessary.

Functions

- `TernaryArrows`: Add arrows
- `TernaryLines`: Add lines
- `TernaryPoints`: Add points
- `TernaryPolygon`: Add polygons
- `TernaryText`: Add text
- `JoinTheDots`: Add points, joined by lines

Author(s)

Martin R. Smith (martin.smith@durham.ac.uk)

Examples

```
coords <- list(
  A = c(1, 0, 2),
  B = c(1, 1, 1),
  C = c(1.5, 1.5, 0),
  D = c(0.5, 1.5, 1)
)
TernaryPlot()
AddToTernary(lines, coords, col='darkgreen', lty='dotted', lwd=3)
TernaryLines(coords, col='darkgreen')
TernaryArrows(coords[1], coords[2:4], col='orange', length=0.2, lwd=1)
TernaryText(coords, cex=0.8, col='red', font=2)
TernaryPoints(coords, pch=1, cex=2, col='blue')
AddToTernary(points, coords, pch=1, cex=3)
```

`cbPalettes` *Palettes compatible with colour blindness*

Description

Colour palettes recommended for use with colour blind audiences.

Usage

```
cbPalette8

cbPalette13

cbPalette15
```

Format

Character vectors of lengths 8, 13 and 15.

Details

cbPalette15 is a Brewer palette. Because colours 4 and 7 are difficult to distinguish from colours 13 and 3, respectively, in individuals with tritanopia, cbPalette13 omits these colours (i.e. cbPalette13 <-cbPalette15[-c(4,7)]).

Source

- cbPalette8: *Wong B. 2011. Color blindness. Nat. Methods. 8:441. doi: 10.1038/nmeth.1618*
- cbPalette15: http://mkweb.bcgsc.ca/biovis2012/color-blindness-palette.png

Examples

```
data('cbPalette8')
plot.new()
plot.window(xlim = c(1, 16), ylim = c(0, 3))
text(1:8 * 2, 3, 1:8, col = cbPalette8)
points(1:8 * 2, rep(2, 8), col = cbPalette8, pch = 15)

data('cbPalette15')
text(1:15, 1, col = cbPalette15)
text(c(4, 7), 1, '[   ]')
points(1:15, rep(0, 15), col = cbPalette15, pch = 15)
```

ColourTernary *Colour a ternary plot according to the output of a function*

Description

Colour a ternary plot according to the output of a function

Usage

```
ColourTernary(values, spectrum = viridisLite::viridis(256L, alpha = 0.6),
  resolution = sqrt(ncol(values)),
  direction = getOption("ternDirection"))

ColorTernary(values, spectrum = viridisLite::viridis(256L, alpha = 0.6),
  resolution = sqrt(ncol(values)),
  direction = getOption("ternDirection"))
```

Arguments

values	Numeric matrix specifying the values associated with each point, generated using TernaryPointValues.
spectrum	Vector of colours to use as a spectrum, or NULL to use values['z',].

`resolution` The number of triangles whose base should lie on the longest axis of the triangle. Higher numbers will result in smaller subdivisions and smoother colour gradients, but at a computational cost.

`direction` (optional) Integer specifying the direction that the current ternary plot should point: 1, up; 2, right; 3, down; 4, left.

Author(s)

Martin R. Smith (martin.smith@durham.ac.uk)

See Also

Other contour plotting functions: `TernaryContour`, `TernaryDensityContour`, `TernaryPointValues`

Other functions for colouring and shading: `TernaryTiles`

Examples

```
TernaryPlot(alab = 'a', blab = 'b', clab = 'c')

FunctionToContour <- function (a, b, c) {
  a - c + (4 * a * b) + (27 * a * b * c)
}

values <- TernaryPointValues(FunctionToContour, resolution = 24L)
ColourTernary(values)
TernaryContour(FunctionToContour, resolution = 36L)


TernaryPlot()
values <- TernaryPointValues(rgb, resolution = 20)
ColourTernary(values, spectrum = NULL)
```

`OutsidePlot` *Is a point in the plotting area?*

Description

Evaluate whether a given set of coordinates lie outwith the boundaries of a plotted ternary diagram.

Usage

```
OutsidePlot(x, y, tolerance = 0)
```

Arguments

`x, y`	Vectors of *x* and *y* coordinates of points.
`tolerance`	Consider points this close to the edge of the plot to be inside. Set to negative values to count points that are just outside the plot as inside, and to positive values to count points that are just inside the margins as outside. Maximum positive value: 1/3.

Value

`OutsidePlot()` returns a logical vector specifying whether each pair of *x* and *y* coordinates corresponds to a point outside the plotted ternary diagram.

Author(s)

Martin R. Smith (martin.smith@durham.ac.uk)

See Also

Other plot limits: `TernaryXRange`

Examples

```
TernaryPlot()
points(0.5, 0.5, col = 'darkgreen')
OutsidePlot(0.5, 0.5)

points(0.1, 0.5, col = 'red')
OutsidePlot(0.1, 0.5)

OutsidePlot(c(0.5, 0.1), 0.5)
```

`ReflectedEquivalents` *Reflected equivalents of points outside the ternary plot*

Description

To avoid edge effects, it may be desirable to add the value of a point within a ternary plot with the value of its 'reflection' across the nearest axis or corner.

Usage

```
ReflectedEquivalents(x, y, direction = getOption("ternDirection"))
```

Arguments

x, y — Vectors of *x* and *y* coordinates of points.

direction — (optional) Integer specifying the direction that the current ternary plot should point: 1, up; 2, right; 3, down; 4, left.

Value

`ReflectedEquivalents()` returns a list of the *x*, *y* coordinates of the points produced if the given point is reflected across each of the edges or corners.

See Also

Other coordinate translation functions: `TernaryCoords`, `TriangleCentres`, `XYToTernary`

Examples

```
TernaryPlot(axis.labels=FALSE, point=4)

xy <- cbind(
  TernaryCoords(0.9, 0.08, 0.02),
  TernaryCoords(0.15, 0.8, 0.05),
  TernaryCoords(0.05, 0.1, 0.85)
)
x <- xy[1, ]
y <- xy[2, ]

points(x, y, col='red', pch=1:3)
ref <- ReflectedEquivalents(x, y)
points(ref[[1]][, 1], ref[[1]][, 2], col='blue', pch=1)
points(ref[[2]][, 1], ref[[2]][, 2], col='green', pch=2)
points(ref[[3]][, 1], ref[[3]][, 2], col='orange', pch=3)
```

TernaryContour — *Add contours to a ternary plot*

Description

Draws contour lines to depict the value of a function in ternary space.

Usage

```
TernaryContour(Func, resolution = 96L,
  direction = getOption("ternDirection"), ...)
```

Arguments

Func	Function taking the parameters a, b and c, which evaluates to a numeric whose value should be depicted.
resolution	The number of triangles whose base should lie on the longest axis of the triangle. Higher numbers will result in smaller subdivisions and smoother colour gradients, but at a computational cost.
direction	(optional) Integer specifying the direction that the current ternary plot should point: 1, up; 2, right; 3, down; 4, left.
...	Further parameters to pass to ‘contour.

Author(s)

Martin R. Smith (martin.smith@durham.ac.uk)

See Also

Other contour plotting functions: `ColourTernary`, `TernaryDensityContour`, `TernaryPointValues`

Examples

```
TernaryPlot(alab = 'a', blab = 'b', clab = 'c')

FunctionToContour <- function (a, b, c) {
  a - c + (4 * a * b) + (27 * a * b * c)
}

values <- TernaryPointValues(FunctionToContour, resolution = 24L)
ColourTernary(values)
TernaryContour(FunctionToContour, resolution = 36L)
```

TernaryCoords *Convert ternary coordinates to Cartesian space*

Description

Convert coordinates of a point in ternary space, in the format (a, b, c), to x and y coordinates of Cartesian space, which can be sent to standard functions in the 'graphics' package.

Usage

```
TernaryCoords(abc, b_coord = NULL, c_coord = NULL,
  direction = getOption("ternDirection"))
```

Arguments

abc	A vector of length three giving the position on a ternary plot that points in the direction specified by `direction` (1 = up, 2 = right, 3 = down, 4 = left). `c(100,0,0)` will plot in the `direction`-most corner; `c(0,100,0)` will plot in the corner clockwise of `direction`; `c(0,0,100)` will plot in the corner anti-clockwise of `direction`. Alternatively, the a coordinate can be specified as the first parameter, in which case the b and c coordinates must be specified via `b_coord` and `c_coord`.
b_coord	The b coordinate, if abc is a single number.
c_coord	The c coordinate, if abc is a single number.
direction	(optional) Integer specifying the direction that the current ternary plot should point: 1, up; 2, right; 3, down; 4, left.

Value

`TernaryCoords()` returns a vector of length two that converts the coordinates given in abc into Cartesian (x, y) coordinates corresponding to the plot created by the last call of `TernaryPlot()`.

Author(s)

Martin R. Smith (martin.smith@durham.ac.uk)

See Also

- `TernaryPlot()`

Other coordinate translation functions: `ReflectedEquivalents`, `TriangleCentres`, `XYToTernary`

Examples

```
TernaryCoords(100, 0, 0)
TernaryCoords(c(0, 100, 0))

coords <- matrix(1:12, ncol=3)
apply(coords, 1, TernaryCoords)
```

`TernaryDensityContour` *Add contours of estimated point density to a ternary plot*

Description

Use two-dimensional kernel density estimation to plot contours of point density.

Usage

```
TernaryDensityContour(coordinates, bandwidth, resolution = 25L,
  tolerance = -0.2/resolution, edgeCorrection = TRUE,
  direction = getOption("ternDirection"), ...)
```

Arguments

`coordinates`	A list, matrix, data.frame or vector in which each element (or row) specifies the three coordinates of a point in ternary space.
`bandwidth`	Vector of bandwidths for x and y directions. Defaults to normal reference bandwidth (see `MASS::bandwidth.nrd`). A scalar value will be taken to apply to both directions.
`resolution`	The number of triangles whose base should lie on the longest axis of the triangle. Higher numbers will result in smaller subdivisions and smoother colour gradients, but at a computational cost.
`tolerance`	Numeric specifying how close to the margins the contours should be plotted, as a fraction of the size of the triangle. Negative values will cause contour lines to extend beyond the margins of the plot.
`edgeCorrection`	Logical specifying whether to correct for edge effects (see details).
`direction`	(optional) Integer specifying the direction that the current ternary plot should point: 1, up; 2, right; 3, down; 4, left.
`...`	Further parameters to pass to ‘contour.

Details

This function is modelled on `MASS::kde2d()`, which uses "an axis-aligned bivariate normal kernel, evaluated on a square grid".

This is to say, values are calculated on a square grid, and contours fitted between these points. This produces a couple of artefacts. Firstly, contours may not extend beyond the outermost point within the diagram, which may fall some distance from the margin of the plot if a low `resolution` is used. Setting a negative `tolerance` parameter allows these contours to extend closer to (or beyond) the margin of the plot.

Individual points cannot fall outside the margins of the ternary diagram, but their associated kernels can. In order to sample regions of the kernels that have 'bled' outside the ternary diagram, each point's value is calculated by summing the point density at that point and at equivalent points outside the ternary diagram, 'reflected' across the margin of the plot (see function `ReflectedEquivalents`). This correction can be disabled by setting the `edgeCorrection` parameter to `FALSE`.

A model based on a triangular grid may be more appropriate in certain situations, but is non-trivial to implement; if this distinction is important to you, please let the maintainers known by opening a Github issue.

Author(s)

Adapted from `MASS::kde2d()` by Martin R. Smith

See Also

Other contour plotting functions: `ColourTernary`, `TernaryContour`, `TernaryPointValues`

Examples

```
TernaryPlot(axis.labels = seq(0, 10, by = 1))

nPoints <- 400L
coordinates <- cbind(abs(rnorm(nPoints, 2, 3)),
                     abs(rnorm(nPoints, 1, 1.5)),
                     abs(rnorm(nPoints, 1, 0.5)))

ColourTernary(TernaryDensity(coordinates, resolution = 10L))
TernaryPoints(coordinates, col = 'red', pch = '.')
TernaryDensityContour(coordinates, resolution = 30L)
```

`TernaryPlot` *Create a ternary plot*

Description

Create and style a blank ternary plot.

Usage

```
TernaryPlot(atip = NULL, btip = NULL, ctip = NULL, alab = NULL,
  blab = NULL, clab = NULL, lab.offset = 0.16, lab.col = NULL,
  point = "up", clockwise = TRUE, xlim = NULL, ylim = NULL,
  lab.cex = 1, lab.font = 0, tip.cex = lab.cex, tip.font = 2,
  tip.col = "black", isometric = TRUE, atip.rotate = NULL,
  btip.rotate = NULL, ctip.rotate = NULL, atip.pos = NULL,
  btip.pos = NULL, ctip.pos = NULL, padding = 0.08, col = NA,
  grid.lines = 10, grid.col = "darkgrey", grid.lty = "solid",
  grid.lwd = par("lwd"), grid.minor.lines = 4,
  grid.minor.col = "lightgrey", grid.minor.lty = "solid",
  grid.minor.lwd = par("lwd"), axis.lty = "solid",
  axis.labels = TRUE, axis.cex = 0.8, axis.font = par("font"),
  axis.tick = TRUE, axis.lwd = 1, ticks.lwd = axis.lwd,
  ticks.length = 0.025, axis.col = "black", ticks.col = grid.col,
  ...)

HorizontalGrid(grid.lines = 10, grid.col = "grey",
  grid.lty = "dotted", grid.lwd = par("lwd"),
  direction = getOption("ternDirection"))
```

Arguments

`atip, btip, ctip`
: Character string specifying text to title corners, proceeding clockwise from the corner specified in `point` (default: top).

`alab, blab, clab`
: Character string specifying text with which to label the corresponding sides of the triangle. Left or right-pointing arrows are produced by typing `\U2190` or `\U2192`, or using `expression('value' %->% '')`.

`lab.offset`
: Numeric specifying distance between midpoint of axis label and the axis. Increase `padding` if labels are being clipped. Use a vector of length three to specify a different offset for each label.

`lab.col`
: Character vector specifying colours for axis labels. Use a vector of length three to specify a different colour for each label.

`point`
: Character string specifying the orientation of the ternary plot: should the triangle point `"up"`, `"right"`, `"down"` or `"left"`? The integers 1 to 4 can be used in place of the character strings.

`clockwise`
: Logical specifying the direction of axes. If `TRUE` (the default), each axis runs from zero to its maximum value in a clockwise direction around the plot.

`xlim, ylim`
: Numeric vectors of length 2 specifying the minimum and maximum x and y limits of the plotted area, to which `padding` will be added. The default is to display the complete height or width of the plot. Allows cropping to magnified region of the plot. (See vignette for diagram.) May be overridden if `isometric=TRUE`; see documentation of `isometric` parameter.

`lab.cex, tip.cex`
: Numeric specifying character expansion for axis labels. Use a vector of length three to specify a different value for each direction.

`lab.font, tip.font`
: Numeric specifying font (Roman, bold, italic, bold-italic) for axis titles. Use a vector of length three to set a different font for each direction.

`isometric`
: Logical specifying whether to enforce an equilateral shape for the ternary plot. If only one of `xlim` and `ylim` is set, the other will be calculated to maintain an equilateral plot. If both `xlim` and `ylim` are set, but have different ranges, then the limit with the smaller range will be scaled until its range matches that of the other limit.

`atip.rotate, btip.rotate, ctip.rotate`
: Integer specifying number of degrees to rotate label of rightmost apex.

`atip.pos, btip.pos, ctip.pos`
: Integer specifying positioning of labels, iff the corresponding `xlab.rotate` parameter is set.

`padding`
: Numeric specifying size of internal margin of the plot; increase if axis labels are being clipped.

`col`
: The colour for filling the plot; see `polygon`.

`grid.lines`
: Integer specifying the number of grid lines to plot.

`grid.col, grid.minor.col`
: Colours to draw the grid lines. Use a vector of length three to set different values for each direction.

`grid.lty, grid.minor.lty`
: Character or integer vector; line type of the grid lines. Use a vector of length three to set different values for each direction.

`grid.lwd, grid.minor.lwd`
: Non-negative numeric giving line width of the grid lines. Use a vector of length three to set different values for each direction.

`grid.minor.lines`
: Integer specifying the number of minor (unlabelled) grid lines to plot between each major pair.

`axis.lty`
: Line type for both the axis line and tick marks. Use a vector of length three to set a different value for each direction.

`axis.labels`
: This can either be a logical value specifying whether (numerical) annotations are to be made at the tickmarks, or a character or expression vector of labels to be placed at the tick points.

`axis.cex`
: Numeric specifying character expansion for axis labels. Use a vector of length three to set a different value for each direction.

`axis.font`
: Font for text. Defaults to `par('font')`.

`axis.tick`
: Logical specifying whether to mark the axes with tick marks.

`axis.lwd, ticks.lwd`
: Line width for the axis line and tick marks. Zero or negative values will suppress the line or ticks. Use a vector of length three to set different values for each axis.

`ticks.length`
: Numeric specifying distance that ticks should extend beyond the plot margin. Also affects position of axis labels, which are plotted at the end of each tick. Use a vector of length three to set a different length for each direction.

`axis.col, ticks.col, tip.col`
: Colours for the axis line, tick marks and tip labels respectively. Use a vector of length three to set a different value for each direction. `axis.col = NULL` means to use `par('fg')`, possibly specified inline, and `ticks.col = NULL` means to use whatever colour `axis.col` resolved to.

`...`
: Additional parameters to `plot`.

`direction`
: (optional) Integer specifying the direction that the current ternary plot should point: 1, up; 2, right; 3, down; 4, left.

Details

The plot will be generated using the standard 'graphics' plot functions, on which additional elements can be added using cartesian coordinates, perhaps using functions such as `arrows`, `legend` or `text`.

Functions

- `HorizontalGrid`: Add `grid.lines` horizontal lines to the ternary plot

Author(s)

Martin R. Smith (martin.smith@durham.ac.uk)

See Also

- `AddToTernary()`: Add elements to a ternary plot
- `TernaryCoords()`: Convert ternary coordinates to Cartesian (*x* and *y*) coordinates
- `TernaryXRange()`, `TernaryYRange()`: What are the *x* and *y* limits of the plotted region?

Examples

```
TernaryPlot(atip="Top", btip="Bottom", ctip="Right", axis.col="red",
            col=rgb(0.8, 0.8, 0.8))
HorizontalGrid(grid.lines=2, grid.col='blue', grid.lty=1)
# the second line corresponds to the base of the triangle, and is not drawn
```

TernaryPointValues — *Value of a function at regularly spaced points*

Description

Intended to facilitate coloured contour plots with `ColourTernary()`, `TernaryPointValue()` evaluates a function at points on a triangular grid; `TernaryDensity()` calculates the density of points in each grid cell.

Usage

```
TernaryPointValues(Func, resolution = 48L,
  direction = getOption("ternDirection"), ...)

TernaryDensity(coordinates, resolution = 48L,
  direction = getOption("ternDirection"))
```

Arguments

`Func`	Function taking the parameters `a`, `b` and `c`, which evaluates to a numeric whose value should be depicted.
`resolution`	The number of triangles whose base should lie on the longest axis of the triangle. Higher numbers will result in smaller subdivisions and smoother colour gradients, but at a computational cost.
`direction`	(optional) Integer specifying the direction that the current ternary plot should point: 1, up; 2, right; 3, down; 4, left.
`...`	Additional parameters to `Func()`.
`coordinates`	A list, matrix, data.frame or vector in which each element (or row) specifies the three coordinates of a point in ternary space.

Value

`TernaryPointValues()` returns a matrix whose rows correspond to:

- **x**, **y**: co-ordinates of the centres of smaller triangles
- **z**: The value of `Func(a,b,c)`, where `a`, `b` and `c` are the ternary coordinates of `x` and `y`.
- **down**: `0` if the triangle concerned points upwards (or right), 1 otherwise

Author(s)

Martin R. Smith (martin.smith@durham.ac.uk)

See Also

Other contour plotting functions: `ColourTernary`, `TernaryContour`, `TernaryDensityContour`

Examples

```
TernaryPointValues(function (a, b, c) a * b * c, resolution = 2)

TernaryPlot(grid.lines = 4)
cols <- TernaryPointValues(rgb, resolution = 4)
text(as.numeric(cols['x', ]), as.numeric(cols['y', ]),
     labels =  ifelse(cols['down', ] == '1', 'v', '^'),
     col = cols['z', ])

TernaryPlot(axis.labels = seq(0, 10, by = 1))

nPoints <- 4000L
coordinates <- cbind(abs(rnorm(nPoints, 2, 3)),
                     abs(rnorm(nPoints, 1, 1.5)),
                     abs(rnorm(nPoints, 1, 0.5)))

density <- TernaryDensity(coordinates, resolution = 10L)
ColourTernary(density)
TernaryPoints(coordinates, col = 'red', pch = '.')
```

`TernaryTiles` *Paint tiles on ternary plot*

Description

Function to fill a ternary plot with coloured tiles. Useful in combination with `TernaryPointValues` and `TernaryContour`.

Usage

```
TernaryTiles(x, y, down, resolution, col,
  direction = getOption("ternDirection"))
```

Arguments

`x, y`	Numeric vectors specifying *x* and *y* coordinates of centres of each triangle.
`down`	Logical vector specifying `TRUE` if each triangle should point down (or right), `FALSE` otherwise.
`resolution`	The number of triangles whose base should lie on the longest axis of the triangle. Higher numbers will result in smaller subdivisions and smoother colour gradients, but at a computational cost.
`col`	Vector specifying the colour with which to fill each triangle.
`direction`	(optional) Integer specifying the direction that the current ternary plot should point: 1, up; 2, right; 3, down; 4, left.

Author(s)

Martin R. Smith (martin.smith@durham.ac.uk)

See Also

Other functions for colouring and shading: `ColourTernary`

Examples

```
FunctionToContour <- function (a, b, c) {
  a - c + (4 * a * b) + (27 * a * b * c)
}

TernaryPlot()

values <- TernaryPointValues(FunctionToContour, resolution = 24L)
ColourTernary(values)
TernaryContour(FunctionToContour, resolution=36L)
```

`TernaryXRange` *X and Y coordinates of ternary plotting area*

Description

X and Y coordinates of ternary plotting area

Usage

```
TernaryXRange(direction = getOption("ternDirection"))

TernaryYRange(direction = getOption("ternDirection"))
```

Arguments

`direction`	(optional) Integer specifying the direction that the current ternary plot should point: 1, up; 2, right; 3, down; 4, left.

Value

`TernaryXRange()` and `TernaryYRange()` return the minimum and maximum X or Y coordinate of the area in which a ternary plot is drawn, oriented in the specified direction. Because the plotting area is a square, the triangle of the ternary plot will not occupy the full range in one direction. Assumes that the defaults have not been overwritten by specifying `xlim` or `ylim`.

Functions

- `TernaryYRange`: Returns the minimum and maximum Y coordinate for a ternary plot in the specified direction.

Author(s)

Martin R. Smith (martin.smith@durham.ac.uk)

See Also

Other plot limits: `OutsidePlot`

`TriangleCentres` *Coordinates of triangle mid-points*

Description

Calculate *x* and *y* coordinates of the midpoints of triangles tiled to cover a ternary plot.

Usage

```
TriangleCentres(resolution = 48L,
  direction = getOption("ternDirection"))
```

Arguments

`resolution`	The number of triangles whose base should lie on the longest axis of the triangle. Higher numbers will result in smaller subdivisions and smoother colour gradients, but at a computational cost.
`direction`	(optional) Integer specifying the direction that the current ternary plot should point: 1, up; 2, right; 3, down; 4, left.

Value

`TriangleCentres()` returns a matrix with three named rows:

- `x` *x* coordinates of triangle midpoints;
- `y` *y* coordinates of triangle midpoints;
- `triDown` `0` for upwards-pointing triangles, `1` for downwards-pointing.

Author(s)

Martin R. Smith (martin.smith@durham.ac.uk)

See Also

Other coordinate translation functions: `ReflectedEquivalents`, `TernaryCoords`, `XYToTernary`

Examples

```
TernaryPlot(grid.lines = 4)
centres <- TriangleCentres(4)
text(centres['x', ], centres['y', ], ifelse(centres['triDown', ], 'v', '^'))
```

`XYToTernary` *Cartesian coordinates to ternary point*

Description

Convert cartesian (x, y) coordinates to a point in ternary space.

Usage

```
XYToTernary(x, y, direction = getOption("ternDirection"))
```

Arguments

`x, y`	Numeric values giving the x and y coordinates of a point or points.
`direction`	(optional) Integer specifying the direction that the current ternary plot should point: 1, up; 2, right; 3, down; 4, left.

Value

`XYToTernary()` Returns the ternary point(s) corresponding to the specified x and y coordinates, where a + b + c = 1.

Author(s)

Martin R. Smith (martin.smith@durham.ac.uk)

See Also

Other coordinate translation functions: ReflectedEquivalents, TernaryCoords, TriangleCentres

Examples

```
XYToTernary(c(0.1, 0.2), 0.5)
```

Index