Package ‘Ternary’

June 29, 2023

Version 2.2.1
Title Create Ternary and Holdridge Plots
Description Plots ternary diagrams (simplex plots / Gibbs triangles) and Holdridge life zone plots <doi:10.1126/science.105.2727.367> using the standard graphics functions.
An alternative to 'ggttern', which uses the 'ggplot2' family of plotting functions.
Includes a 'Shiny' user interface for point-and-click ternary plotting.

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 (>= 3.5.0)
Imports RcppHungarian, PlotTools (>= 0.2.0), shiny, sp, viridisLite,
Suggests colourpicker, knitr, readxl, rmarkdown, shinyjs, spelling,
testthat (>= 3.0), vdiffr,
Config/Needs/check rcmdcheck
Config/Needs/coverage covr
Config/Needs/memcheck devtools, rcmdcheck
Config/Needs/metadata codemeta
Config/Needs/revdeps revdepcheck
Config/Needs/website pkgdown
Config/testthat/edition 3
Config/testthat/parallel false
LazyData true
VignetteBuilder knitr
Encoding UTF-8
RoxygenNote 7.2.3
AddToHoldridge

Add elements to ternary or Holdridge plot

Description

Plot shapes onto a ternary diagram created with `TernaryPlot()`, or a Holdridge plot created with `HoldridgePlot()`.
AddToHoldridge

Usage

AddToHoldridge(PlottingFunction, pet, prec, ...)

HoldridgeArrows(fromCoordinates, toCoordinates = fromCoordinates, ...)

HoldridgeLines(pet, prec, ...)

HoldridgePoints(pet, prec, ...)

HoldridgePolygon(pet, prec, ...)

HoldridgeText(pet, prec, ...)

AddToTernary(PlottingFunction, coordinates, ...)

TernarySegments(fromCoordinates, toCoordinates = fromCoordinates, ...)

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.

pet, prec Numeric vectors giving potential evapotranspiration ratio and annual precipitation (in mm).

... Additional parameters to pass to PlottingFunction(). If using TernaryText(), this will likely include the parameter labels, to specify the text to plot. Other useful graphical parameters include srt to rotate text.

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

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

Functions

• HoldridgeArrows(): Add arrows to Holdridge plot
- HoldridgeLines(): Add lines to Holdridge plot
- HoldridgePoints(): Add points to Holdridge plot
- HoldridgePolygon(): Add polygons to Holdridge plot
- HoldridgeText(): Add text to Holdridge plot
- TernarySegments(): Add segments
- 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)

**See Also**

Other Holdridge plotting functions: `HoldridgeHypsometricCol()`, `HoldridgePlot()`, `holdridgeClasses`, `holdridge`

**Examples**

```r
# Data to plot
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)
)

# Set up plot
oPar <- par(mar = rep(0, 4), xpd = NA) # reduce margins and write in them
TernaryPlot()

# Add elements to ternary diagram
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(graphics::points, coords, pch = 1, cex = 3)

# An equivalent syntax applies to Holdridge plots:
HoldridgePlot()
pet <- c(0.8, 2, 0.42)
prec <- c(250, 400, 1337)
HoldridgeText(pet, prec, c("A", "B", "C"))
```
Annotate

AddToHoldridge(graphics::points, pet, prec, cex = 3)

# Restore original plotting parameters
par(oPar)

Annotate points on a ternary plot

Description

Annotate() identifies and label individual points on a ternary diagram in the plot margins.

Usage

Annotate(
  coordinates,
  labels,
  side,
  outset = 0.16,
  line.col = col,
  lty = par("lty"),
  lwd = par("lwd"),
  col = par("col"),
  font = par("font"),
  offset = 0.5,
  ...
)

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.
labels             Character vector specifying text with which to annotate each entry in coordinates.
side               Optional vector specifying which side of the ternary plot each point should be labelled on, using the notation "a", "b", "c" or 1, 2, 3. Entries of "n" or 0 will not be annotated (but still require an entry in labels). Entries of NA will be allocated a side automatically, based on the midpoint of coordinates.
outset             Numeric specifying distance from plot margins to labels.
line.col, lty, lwd  parameters to segments().
col, font, offset   parameters to text().
...                 Further parameters to text() and segments().

Author(s)

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

Annotation vignette gives further suggestions for manual annotation.

Examples

```r
# Load some data
data("Seatbelts")
seats <- c("drivers", "front", "rear")
seat <- Seatbelts[month.abb %in% "Oct", seats]
law <- Seatbelts[month.abb %in% "Oct", "law"]

# Set up plot
oPar <- par(mar = c(2, 0, 0, 0))
TernaryPlot(alab = seats[1], blab = seats[2], clab = seats[3])
TernaryPoints(seat, cex = 0.8, col = 2 + law)

# Annotate points by year
Annotate(seat, labels = 1969:1984, col = 2 + law)

# Restore original graphical parameters
par(oPar)
```

---

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.

An object of class character of length 8.
An object of class character of length 13.
An object of class character of length 15.
Details
Since R 4.0, cbPalette8 is available in base R as palette.colors(8).
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
• cbPalette15: http://mkweb.bcgsc.ca/biovis2012/color-blindness-palette.png

Examples
```r
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)
```

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

Usage
```r
ColourTernary(
  values,
  spectrum = viridisLite::viridis(256L, alpha = 0.6),
  resolution = sqrt(ncol(values)),
  direction =getOption("ternDirection", 1L),
  legend,
  ...
)
```
```r
ColorTernary(
  values,
  spectrum = viridisLite::viridis(256L, alpha = 0.6),
  resolution = sqrt(ncol(values)),
  direction =getOption("ternDirection", 1L),
)```
Arguments

values Numeric matrix, possibly created using `TernaryPointValues()`, with four named rows: `x, y`, cartesian coordinates of each triangle centre; `z`, value associated with that coordinate; `down`, triangle direction: `0` = point upwards; `1` = point downwards.

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.

legend Character vector specifying annotations for colour scale. If not provided, no colour legend is displayed. Specify `TRUE` to generate automatically, or a single integer to generate legend annotations.

Further arguments to `SpectrumLegend()`.

Author(s)

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

See Also

Fine control over continuous legends: `PlotTools::SpectrumLegend()`

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

Other functions for colouring and shading: `TernaryTiles()`

Examples

```r
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,
  x = "topleft",
  bty = "n", # No box
  legend = signif(seq(max(values), min(values), length.out = 4), 3)
)
TernaryContour(FunctionToContour, resolution = 36L)
```
TernaryPlot()
values <- TernaryPointValues(rgb, resolution = 20)
ColourTernary(values, spectrum = NULL)

# Create a helper function to place white centrally:
rgbWhite <- function (r, g, b) {
  highest <- apply(rbind(r, g, b), 2L, max)
  rgb(r/highest, g/highest, b/highest)
}
TernaryPlot()
values <- TernaryPointValues(rgbWhite, resolution = 20)
ColourTernary(values, spectrum = NULL)

---

holdridge

**Random sample of points for Holdridge plotting**

**Description**
A stratified random sampling (average of 100 points) using a global mapping of Holdridge's scheme.

**Usage**
holdridge

**Format**
An object of class data.frame with 39 rows and 4 columns.

**Author(s)**
James Lee Tsakalos

**See Also**
Other Holdridge plotting functions: `AddToHoldridge()`, `HoldridgeHypsometricCol()`, `HoldridgePlot()`, `holdridgeClasses`

**Examples**
```r
data("holdridge", package = "Ternary")
head(holdridge)
```
### holdridgeClasses

*Names of the 38 classes defined with the Holdridge system*

#### Description

`holdridgeClasses` is a character vector naming, from left to right, top to bottom, the 38 classes defined by the International Institute for Applied Systems Analysis (IIASA).

#### Usage

- `holdridgeClasses`
- `holdridgeLifeZones`
- `holdridgeLifeZonesUp`
- `holdridgeClassesUp`

#### Format

- An object of class character of length 38.
- An object of class character of length 33.
- An object of class character of length 33.
- An object of class character of length 38.

#### Details

- `holdridgeLifeZones` is a character vector naming, from left to right, top to bottom, the 38 cells of the Holdridge classification plot.
- `holdridgeClassesUp` and `holdridgeLifeZonesUp` replace spaces with new lines, for more legible plotting with `HoldridgeHexagons()`.

#### Author(s)

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

#### Source

- Holdridge (1967), *Life zone ecology*. Tropical Science Center, San José
HoldridgeHypsometricCol

Convert a point in evapotranspiration-precipitation space to an appropriate cross-blended hypsometric colour

Description

Used to colour HoldridgeHexagons(), and may also be used to aid the interpretation of PET + precipitation data in any graphical context.

Usage

HoldridgeHypsometricCol(pet, prec, opacity = NA)

Arguments

- **pet, prec**: Numeric vectors giving potential evapotranspiration ratio and annual precipitation (in mm).
- **opacity**: Opacity level to be converted to the final two characters of an RGBA hexadecimal colour definition, e.g. #000000FF. Specify a character string, which will be interpreted as a hexadecimal alpha value and appended to the six RGB hexadecimal digits; a numeric in the range 0 (transparent) to 1 (opaque); or NA, to return only the six RGB digits.

Value

Character vector listing RGB or (if opacity != NA) RGBA values corresponding to each PET-precipitation value pair.

Author(s)

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

References

Palette derived from the hypsometric colour scheme presented at Shaded Relief.

See Also

Other Holdridge plotting functions: AddToHoldridge(), HoldridgePlot(), holdridgeClasses, holdridge
Examples

```r
HoldridgePlot(hex.col = HoldridgeHypsometricCol)
VeryTransparent <- function(...) HoldridgeHypsometricCol(..., opacity = 0.3)
HoldridgePlot(hex.col = VeryTransparent)
pet <- holdridge$PET
prec <- holdridge$Precipitation
ptCol <- HoldridgeHypsometricCol(pet, prec)
HoldridgePoints(pet, prec, pch = 21, bg = ptCol)
```

Description

`HoldridgePlot()` creates a blank triangular plot, as proposed by Holdridge (1947, 1967), onto which potential evapotranspiration (PET) ratio and annual precipitation data can be plotted (using the `AddToHoldridge()` family of functions) in order to interpret climatic life zones.

Usage

```r
HoldridgePlot(
  atip = NULL,
  btip = NULL,
  ctip = NULL,
  alab = "Potential evapotranspiration ratio",
  blab = "Annual precipitation / mm",
  clab = "Humidity province",
  lab.offset = 0.22,
  lab.col = c("#D81B60", "#1E88E5", "#111111"),
  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 = 0.16,
  col = NA,
  panel.first = NULL,
  panel.last = NULL,
)```
grid.lines = 8,
grid.col = c(NA, "#1E88E5", "#D81B60"),
grid.lty = "solid",
grid.lwd = par("lwd"),
grid.minor.lines = 0,
grid.minor.col = "lightgrey",
grid.minor.lty = "solid",
grid.minor.lwd = par("lwd"),
hex.border = "#888888",
hex.col = HoldridgeHypsometricCol,
hex.lty = "solid",
hex.lwd = par("lwd"),
hex.cex = 0.5,
hex.labels = NULL,
hex.font = NULL,
hex.text.col = "black",
axis.cex = 0.8,
axis.col = c(grid.col[2], grid.col[3], NA),
axis.font = par("font"),
axis.labels = TRUE,
axis.lty = "solid",
axis.lwd = 1,
axis.rotate = TRUE,
axis.pos = NULL,
axis.tick = TRUE,
ticks.lwd = axis.lwd,
ticks.length = 0.025,
ticks.col = grid.col,

HoldridgeBelts(
    grid.col = "#004D40",
    grid.lty = "dotted",
    grid.lwd = par("lwd")
)

HoldridgeHexagons(
    border = "#004D40",
    hex.col = HoldridgeHypsometricCol,
    lty = "dotted",
    lwd = par("lwd"),
    labels = NULL,
    cex = 1,
    text.col = NULL,
    font = NULL
)
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.

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 (font size) for axis labels. Use a vector of length three to specify a different value for each direction.

lab.font, tip.font  Numeric specifying font style (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.

panel.first  An expression to be evaluated after the plot axes are set up but before any plotting takes place. This can be useful for drawing backgrounds, e.g. with ColourTernary() or HorizontalGrid(). Note that this works by lazy evaluation: passing this argument from other plot methods may well not work since it may be evaluated too early.

panel.last  An expression to be evaluated after plotting has taken place but before the axes and box are added. See the comments about panel.first.

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.

hex.border, hex.lty, hex.lwd
   Parameters to pass to HoldridgeHexagons(). Set to NA to suppress hexagons.

hex.col
   Fill colour for hexagons. Provide a vector specifying a colour for each hexagon
   in turn, reading from left to right and top to bottom, or a function that accepts two
   arguments, numerics pet and prec, and returns a colour in a format accepted by
   polygon().

hex.cex, hex.font, hex.text.col
   Parameters passed to text() to plot hex.labels.

hex.labels
   38-element character vector specifying label for each hexagonal class, from top
   left to bottom right.

axis.cex
   Numeric specifying character expansion (font size) for axis labels. Use a vector
   of length three to set a different value 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.

axis.font
   Font for text. Defaults to par("font").

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, or a list of length three, with each entry specifying
   labels to be placed on each axis in turn.

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.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.

axis.rotate
   Logical specifying whether to rotate axis labels to parallel grid lines, or numeric
   specifying custom rotation for each axis, to be passed as srt parameter to
   text(). Expand margins or set par(xpd = NA) if labels are clipped.

axis.pos
   Vector of length one or three specifying position of axis labels, to be passed as
   pos parameter to text(); populated automatically if NULL (the default).

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

... Additional parameters to `plot`.

border: Colour to use for hexagon borders.

lty, lwd, cex, font: Graphical parameters specifying properties of hexagons to be plotted.

labels: Vector specifying labels for life zone hexagons to be plotted. Suggested values: `holdridgeClassesUp`, `holdridgeLifeZonesUp`.

text.col: Colour of text to be printed in hexagons.

Details

`HoldridgePoints()`, `HoldridgeText()` and related functions allow data points to be added to an existing plot; `AddToHoldridge()` allows plotting using any of the standard plotting functions.

`HoldridgeBelts()` and `HoldridgeHexagons()` plot interpretative lines and hexagons allowing plotted data to be linked to interpreted climate settings.

Author(s)

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

References


Holdridge (1967), *Life zone ecology*. Tropical Science Center, San José

See Also

Other Holdridge plotting functions: `AddToHoldridge()`, `HoldridgeHypsometricCol()`, `holdridgeClasses`, `holdridge`

Examples

data(holdridgeLifeZonesUp, package = "Ternary")
HoldridgePlot(hex.labels = holdridgeLifeZonesUp)
HoldridgeBelts()
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)
Description

Geometry functions for irregular polygons.

Usage

PolygonArea(x, y = NULL, positive = TRUE)

PolygonCentre(x, y = NULL)

PolygonCenter(x, y = NULL)

GrowPolygon(x, y = NULL, buffer = 0)

Arguments

x, y  Vectors containing the coordinates of the vertices of the polygon.
positive  If vertices are specified in an anticlockwise direction, the polygon will be treated as a hole, with a negative area, unless positive is set to TRUE. Vertices specified in a clockwise sequence always yield a positive area.
buffer  Numeric specifying distance by which to grow polygon.

Value

PolygonArea() returns the area of the specified polygon.
PolygonCentre() returns a single-row matrix containing the x and y coordinates of the geometric centre of the polygon.
GrowPolygon() returns coordinates of the vertices of polygon after moving each vertex buffer away from the polygon’s centre.

Functions

- PolygonArea(): Calculate the area of an irregular polygon
- PolygonCentre(): Locate the centre of a polygon
- GrowPolygon(): Enlarge a polygon in all directions

Author(s)

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

See Also

Other tiling functions: TriangleCentres(), TriangleInHull()
ReflectedEquivalents

Examples

```r
x <- c(-3, -1, 6, 3, -4)
y <- c(-2, 4, 1, 10, 9)
plot(x, y, frame.plot = FALSE)
polygon(x, y)
PolygonArea(x, y)
points(PolygonCentre(x, y), pch = 3, cex = 2)
polygon(GrowPolygon(x, y, 1), border = "darkgreen",
       xpd = NA) # Allow drawing beyond plot border

# Negative values shrink the polygon
polygon(GrowPolygon(x, y, -1), border = "red")
```

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", 1L))`

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

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

xy <- cbind(
    TernaryCoords(0.9, 0.08, 0.02),
    TernaryCoords(0.15, 0.8, 0.05),
    ReflectedEquivalents(0.9, 0.08, 0.02),
    ReflectedEquivalents(0.15, 0.8, 0.05),
)
```
TernaryApp

Graphical user interface for creating ternary plots

Description

TernaryApp() launches a ‘Shiny’ application for the construction of ternary plots. The ‘app’ allows data to be loaded and plotted, and provides code to reproduce the plot in R should more sophisticated plotting functions be desired.

Usage

TernaryApp()

Details

Load data:
The ‘Load data’ input tab allows for the upload of datasets. Data can be read from csv files, .txt files created with write.table(), or (if the ‘readxl’ package is installed) Excel spreadsheets.

Data should be provided as three columns, corresponding to the three axes of the ternary plot. Colours or point styles may be specified in columns four to six to allow different categories of point to be plotted distinctly. Example datasets are installed at system.file("TernaryApp", package = "Ternary").

Axes are automatically labelled using column names, if present; these can be edited manually on this tab.

Plot display:
Allows the orientation, colour and configuration of the plot and its axes to be adjusted.

Grids:
Adjust the number, spacing and styling of major and minor grid lines.

Labels:
Configure the colour, position and size of tip and axis labels.

Points:
Choose whether to plot points, lines, connected points, or text. Set the style of points and lines.
Exporting plots

A plot can be saved to PDF or as a PNG bitmap at a specified size. Alternatively, R script that will generate the displayed plot can be viewed (using the ’R code’ output tab) or downloaded to file.

Author(s)

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

References

If you use figures produced with this package in a publication, please cite


See Also

Full detail of plotting with ’Ternary’, including features not (yet) implemented in the application, is provided in the accompanying vignette.

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", 1L),
  within = NULL,
  filled = FALSE,
  legend,
  legend... = list(),
  nlevels = 10,
  levels = pretty(zlim, nlevels),
  zlim,
  color.palette = function(n) viridisLite::viridis(n, alpha = 0.6),
  fill.col = color.palette(length(levels) - 1),
  ...
)


TernaryContour

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Func</td>
<td>Function taking vectors of coordinates a, b, and c, which returns a numeric vector whose value at each coordinate will be depicted.</td>
</tr>
<tr>
<td>resolution</td>
<td>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.</td>
</tr>
<tr>
<td>direction</td>
<td>(optional) Integer specifying the direction that the current ternary plot should point: 1, up; 2, right; 3, down; 4, left.</td>
</tr>
<tr>
<td>within</td>
<td>List or matrix of x, y coordinates within which contours should be evaluated, in any format supported by <code>xy.coords(x=within)</code>. If NULL, defaults to a region slightly smaller than the ternary plot. The <code>$hull</code> entry generated by <code>TriangleInHull()</code> may also be used.</td>
</tr>
<tr>
<td>filled</td>
<td>Logical; if TRUE, contours will be filled (using <code>.filled.contour()</code>).</td>
</tr>
<tr>
<td>legend</td>
<td>Character vector specifying annotations for colour scale. If not provided, no colour legend is displayed. Specify TRUE to generate automatically, or a single integer to generate legend annotations.</td>
</tr>
<tr>
<td>legend...</td>
<td>List of additional parameters to send to <code>SpectrumLegend()</code>.</td>
</tr>
<tr>
<td>nlevels, levels, zlim,...</td>
<td>parameters to pass to <code>contour()</code>.</td>
</tr>
<tr>
<td>color.palette</td>
<td>parameters to pass to <code>.filled.contour()</code>.</td>
</tr>
<tr>
<td>fill.col</td>
<td>Sent as <code>col</code> parameter to <code>.filled.contour()</code>. Computed from <code>color.palette</code> if not specified.</td>
</tr>
</tbody>
</table>

Value

`TernaryContour()` invisibly returns a list containing:

- `x, y`: the Cartesian coordinates of each evaluated point;
- `z`: The value of `Func()` at each coordinate.

Author(s)

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

See Also

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

Examples

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

# Set up plot
originalPar <- par(mar = rep(0, 4))
TernaryPlot(alab = "a", blab = "b", clab = "c")
```
```
values <- TernaryPointValues(FunctionToContour, resolution = 24L)
ColourTernary(values,
    legend = signif(seq(max(values), min(values), length.out = 4), 2),
    bty = "n")
TernaryContour(FunctionToContour, resolution = 36L)

# Note that FunctionToContour is sent a vector.
# Instead of
BadMax <- function (a, b, c) {
    max(a, b, c)
}

# Use
GoodMax <- function (a, b, c) {
    pmax(a, b, c)
}
TernaryPlot(alab = "a", blab = "b", clab = "c")
ColourTernary(TernaryPointValues(GoodMax))
TernaryContour(GoodMax)

# Or, for a generalizable example,
GeneralMax <- function (a, b, c) {
    apply(rbind(a, b, c), 2, max)
}
TernaryPlot(alab = "a", blab = "b", clab = "c")
# Fill the contour areas, rather than using tiles
TernaryContour(GeneralMax, filled = TRUE,
    legend = c("Max", "Min"), bty = "n",
    fill.col = viridisLite::viridis(14, alpha = 0.6))
# Re-draw edges of plot triangle over fill
TernaryPolygon(diag(3))
# Restore plotting parameters
par(originalPar)
```

---

**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**

```r
TernaryCoords(
    abc,
    b_coord = NULL,
    c_coord = NULL,
    direction = getOption("ternDirection", 1L)
)```
## S3 method for class 'matrix'
TernaryToXY(
  abc,
  b_coord = NULL,
  c_coord = NULL,
  direction = getOption("ternDirection", 1L)
)

## S3 method for class 'numeric'
TernaryToXY(
  abc,
  b_coord = NULL,
  c_coord = NULL,
  direction = getOption("ternDirection", 1L)
)

TernaryToXY(
  abc,
  b_coord = NULL,
  c_coord = NULL,
  direction = getOption("ternDirection", 1L)
)

### 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`. Or, a matrix with three rows, representing in turn the a, b and c coordinates of points.

- **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

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

coops <- matrix(1:12, nrow = 3)
TernaryToXY(coops)
```

---

### 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**

```r
TernaryDensityContour(
  coordinates,
  bandwidth,
  resolution = 25L,
  tolerance = -0.2/resolution,
  edgeCorrection = TRUE,
  direction =getOption("ternDirection", 1L),
  filled = FALSE,
  nlevels = 10,
  levels = pretty(zlim, nlevels),
  zlim,
  color.palette = function(n) viridisLite::viridis(n, alpha = 0.6),
  fill.col = color.palette(length(levels) - 1),
  ...
)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>coordinates</td>
<td>A list, matrix, data.frame or vector in which each element (or row) specifies the three coordinates of a point in ternary space.</td>
</tr>
<tr>
<td>bandwidth</td>
<td>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.</td>
</tr>
<tr>
<td>resolution</td>
<td>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.</td>
</tr>
</tbody>
</table>
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.

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

filled Logical; if TRUE, contours will be filled (using \texttt{.filled.contour()}).

nlevels, levels, zlim, ... parameters to pass to \texttt{contour()}. 

color.palette parameters to pass to \texttt{.filled.contour()}. 

fill.col Sent as \texttt{col} parameter to \texttt{.filled.contour()}. Computed from \texttt{color.palette} if not specified.

Details

This function is modelled on \texttt{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 \texttt{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 \texttt{ReflectedEquivalents}). This correction can be disabled by setting the \texttt{edgeCorrection} parameter to \texttt{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.

Value

\texttt{TernaryDensityContour()} invisibly returns a list containing:

- \texttt{x,y}: the Cartesian coordinates of each grid coordinate;
- \texttt{z}: The density at each grid coordinate.

Author(s)

Adapted from \texttt{MASS::kde2d()} by Martin R. Smith

See Also

Other contour plotting functions: \texttt{ColourTernary()}, \texttt{TernaryContour()}, \texttt{TernaryPointValues()}
TernaryPlot

Examples

# Generate some example data
nPoints <- 400L
coordinates <- cbind(abs(rnorm(nPoints, 2, 3)),
                     abs(rnorm(nPoints, 1, 1.5)),
                     abs(rnorm(nPoints, 1, 0.5)))

# Set up plot
doPar <- par(mar = rep(0, 4))
TernaryPlot(axis.labels = seq(0, 10, by = 1))

# Colour background by density
ColourTernary(TernaryDensity(coordinates, resolution = 10L),
               legend = TRUE, bty = "n", title = "Density")

# Plot points
TernaryPoints(coordinates, col = "red", pch = ".")

# Contour by density
TernaryDensityContour(coordinates, resolution = 30L)

# Reset plotting parameters
par(doPar)

TernaryPlot  Create a ternary plot

Description

Create and style a blank ternary plot.

Usage

TernaryPlot(
    atip = NULL,  # Tip for A
    btip = NULL,  # Tip for B
    ctip = NULL,  # Tip for C
    alab = NULL,  # Label for A
    blab = NULL,  # Label for B
    clab = NULL,  # Label for C
    lab.offset = 0.16,  # Offset for labels
    lab.col = NULL,  # Color for labels
    point = "up",  # Point style
    clockwise = TRUE,  # Clockwise
    xlim = NULL,  # X limits
    ylim = NULL,  # Y limits
    lab.cex = 1,  # Label size
    lab.font = 0,  # Label font
    tip.cex = lab.cex)
TernaryPlot

```r
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, 
panel.first = NULL, 
panel.last = NULL, 
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.rotate = TRUE, 
axis.pos = NULL, 
axis.tick = TRUE, 
axis.lwd = 1, 
ticks.lwd = axis.lwd, 
ticks.length = 0.025, 
axis.col = "black", 
ticks.col = grid.col, 
```

```r
)
```

```r
HorizontalGrid( 
  grid.lines = 10, 
  grid.col = "grey", 
  grid.lty = "dotted", 
  grid.lwd = par("lwd"), 
  direction = getOption("ternDirection", 1L) 
)
```

**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' %->% 'value').

lab.offset Numeric specifying distance between midpoint of axis label and the axis. In-
crease padding if labels are being clipped. Use a vector of length three to spec-
ify 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 (font size) for axis labels. Use a vector
of length three to specify a different value for each direction.

lab.font, tip.font Numeric specifying font style (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 pa-
rameter 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.

panel.first An expression to be evaluated after the plot axes are set up but before any
plotting takes place. This can be useful for drawing backgrounds, e.g. with
ColourTernary() or HorizontalGrid(). Note that this works by lazy eval-
uation: passing this argument from other plot methods may well not work since it
may be evaluated too early.

panel.last An expression to be evaluated after plotting has taken place but before the axes
and box are added. See the comments about panel.first.
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, or a list of length three, with each entry specifying labels to be placed on each axis in turn.
axis.cex    Numeric specifying character expansion (font size) 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.rotate
            Logical specifying whether to rotate axis labels to parallel grid lines, or numeric specifying custom rotation for each axis, to be passed as srt parameter to text(). Expand margins or set par(xpd = NA) if labels are clipped.
axis.pos
            Vector of length one or three specifying position of axis labels, to be passed as pos parameter to text(); populated automatically if NULL (the default).
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

```r
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

```r
TernaryPointValues(
  Func,
  resolution = 48L,
  direction = getOption("ternDirection", 1L),
  ...
)
```
coordinates,
resolution = 48L,
direction = getOption("ternDirection", 1L)
)

Arguments

Func Function taking vectors of coordinates a, b and c, which returns a numeric vector whose value at each coordinate will 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, 1.5)),
                     abs(rnorm(nPoints, 1, 1.5)),
                     abs(rnorm(nPoints, 1, 1.5)),
                     abs(rnorm(nPoints, 1, 1.5)),
                     abs(rnorm(nPoints, 1, 1.5)),
                     abs(rnorm(nPoints, 1, 1.5)),
                     abs(rnorm(nPoints, 1, 1.5)),
                     abs(rnorm(nPoints, 1, 1.5)))
abs(rnorm(nPoints, 1, 0.5)))

density <- TernaryDensity(coordinates, resolution = 10L)
ColourTernary(density, legend = TRUE, bty = "n", title = "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**

```r
TernaryTiles(
  x,
  y,
  down,
  resolution,
  col,
  direction = getOption("ternDirection", 1L)
)
```

**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

TernaryPlot()
TernaryXRange()
TernaryYRange()

TernaryTiles(0, 0.5, TRUE, 10, "red")
xy <- TernaryCoords(c(4, 3, 3))
TernaryTiles(xy[1], xy[2], FALSE, 5, "darkblue")

| TernaryXRange | X and Y coordinates of ternary plotting area |

Description

X and Y coordinates of ternary plotting area

Usage

TernaryXRange(direction = getOption("ternDirection", 1L))
TernaryYRange(direction = getOption("ternDirection", 1L))

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", 1L))

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 coordinates of triangle midpoints;
- 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

Add triangles to a plot: TernaryTiles()
Other coordinate translation functions: ReflectedEquivalents(), TernaryCoords(), XYToTernary()
Other tiling functions: Polygon-Geometry, TriangleInHull()

Examples

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

Does triangle overlap convex hull of points?

Description

Does triangle overlap convex hull of points?

Usage

TriangleInHull(triangles, coordinates, buffer)

Arguments

triangles Three-row matrix as produced by TriangleCentres().
coordinates A matrix with two or three rows specifying the coordinates of points in x, y or a, b, c format.
buffer Include triangles whose centres lie within buffer triangles widths (i.e. edge lengths) of the convex hull.

Value

TriangleInHull() returns a list with the elements:

• $inside: vector specifying whether each of a set of triangles produced by TriangleCentres() overlaps the convex hull of points specified by coordinates.
• $hull: Coordinates of convex hull of coordinates, after expansion to cover overlapping triangles.

Author(s)

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

See Also

Other tiling functions: Polygon-Geometry, TriangleCentres()

Examples

set.seed(0)
nPts <- 50
a <- runif(nPts, 0.3, 0.7)
b <- 0.15 + runif(nPts, 0, 0.7 - a)
c <- 1 - a - b
coordinates <- rbind(a, b, c)

TernaryPlot(grid.lines = 5)
TernaryPoints(coordinates, pch = 3, col = 4)
triangles <- TriangleCentres(resolution = 5)
XYToTernary <- TriangleInHull(triangles, coordinates)
polygon(inHull$hull, border = 4)
values <- rbind(triangles,
  z = ifelse(inHull$inside, "#33cc3333", "#cc333333"))
points(triangles["x", ], triangles["y", ],
  pch = ifelse(triangles["triDown", ], 6, 2),
  col = ifelse(inHull$inside, "#33cc33", "#cc3333"))
ColourTernary(values)

---

**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", 1L))

XYToHoldridge(x, y)

XYToPetPrec(x, y)

**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

* Holdridge plotting functions
  AddToHoldridge, 2
  holdridge, 9
  holdridgeClasses, 10
  HoldridgeHypsometricCol, 11
  HoldridgePlot, 12
* contour plotting functions
  ColourTernary, 7
  TernaryContour, 21
  TernaryDensityContour, 25
  TernaryPointValues, 31
* coordinate translation functions
  ReflectedEquivalents, 19
  TernaryCoords, 23
  TriangleCentres, 35
  XYToTernary, 37
* datasets
  cbPalettes, 6
  holdridge, 9
  holdridgeClasses, 10
* functions for colouring and shading
  ColourTernary, 7
  TernaryTiles, 33
* plot limits
  OutsidePlot, 17
  TernaryXRange, 34
* tiling functions
  Polygon-Geometry, 18
  TriangleCentres, 35
  TriangleInHull, 36
  .filled.contour, 22, 26
AddToHoldridge, 2, 9, 11, 16
AddToHoldridge(), 12, 16
AddToTernary(AddToHoldridge), 2
AddToTernary(), 31
Annotate, 5
arrows, 3, 4, 31
cbPalette13 (cbPalettes), 6
cbPalette15 (cbPalettes), 6
cbPalette8 (cbPalettes), 6
cbPalettes, 6
ColorTernary (ColourTernary), 7
ColourTernary, 7, 22, 26, 32, 33
ColourTernary(), 14, 29, 31
contour, 22, 26
Graphical parameters, 16
graphical parameters, 3
GrowPolygon (Polygon-Geometry), 18
holdridge, 4, 9, 11, 16
HoldridgeArrows (AddToHoldridge), 2
HoldridgeBelts (HoldridgePlot), 12
HoldridgeBelts(), 16
holdridgeClasses, 4, 9, 10, 11, 16
holdridgeClassesUp, 16
holdridgeClassesUp (holdridgeClasses), 10
HoldridgeHexagons (HoldridgePlot), 12
HoldridgeHexagons(), 10, 16
HoldridgeHypsometricCol, 4, 9, 11, 11, 16
holdridgeLifeZones (holdridgeClasses), 10
holdridgeLifeZonesUp, 16
holdridgeLifeZonesUp (holdridgeClasses), 10
HoldridgeLines (AddToHoldridge), 2
HoldridgePlot, 4, 9, 11, 12
HoldridgePlot(), 2
HoldridgePoints (AddToHoldridge), 2
HoldridgePoints(), 16
HoldridgePolygon (AddToHoldridge), 2
HoldridgeText (AddToHoldridge), 2
HoldridgeText(), 16
HorizontalGrid (TernaryPlot), 27
HorizontalGrid(), 14, 29
JoinTheDots (AddToHoldridge), 2