# Package 'metR'

February 15, 2022

**Type** Package

**Language** en-GB

**Title** Tools for Easier Analysis of Meteorological Fields

**Version** 0.12.0

**Description** Many useful functions and extensions for dealing with meteorological data in the tidy data framework. Extends 'ggplot2' for better plotting of scalar and vector fields and provides commonly used analysis methods in the atmospheric sciences.

**License** GPL-3

**URL** https://github.com/eliocamp/metR

**BugReports** https://github.com/eliocamp/metR/issues

**Depends** R (>= 2.10)

**Imports** checkmate, data.table, digest, fields, Formula, formula.tools, ggplot2 (>= 3.0.0), grid, gtable, memoise, plyr, scales, sp, stringr, purrr, RCurl, isoband, lubridate

**Suggests** maps, maptools, covr, irlba, knitr, ncdf4, pkgdown, reshape2, rmarkdown, testthat (>= 2.1.0), viridis, udunits2, PCICt, gridExtra, vdiffr, proj4, kriging, raster, rgdal, here

**ByteCompile** yes

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 7.1.2

**VignetteBuilder** knitr

**NeedsCompilation** no

**Author** Elio Campitelli [cre, aut] (<https://orcid.org/0000-0002-7742-9230>)

**Maintainer** Elio Campitelli <elio.campitelli@cima.fcen.uba.ar>

**Repository** CRAN

**Date/Publication** 2022-02-15 11:00:02 UTC

# R topics documented:

---

`Anomaly` *Anomalies*

---

## Description

Saves keystrokes for computing anomalies.

## Usage

```
Anomaly(x, baseline = seq_along(x), ...)
```

## Arguments

| | |
|---|---|
| `x` | numeric vector |
| `baseline` | logical or numerical vector used for subsetting x before computing the mean |
| `...` | other arguments passed to `mean` such as `na.rm` |

## Value

A numeric vector of the same length as x with each value's distance to the mean.

## See Also

Other utilities: `JumpBy()`, `Mag()`, `Percentile()`, `logic`

## Examples

```
# Zonal temperature anomaly
library(data.table)
temperature[, .(lon = lon, air.z = Anomaly(air)), by = .(lat, lev)]
```

## as.discretised_scale *Create discretised versions of continuous scales*

### Description

Create discretised versions of continuous scales

### Usage

```
as.discretised_scale(scale_function)
```

### Arguments

| | |
|---|---|
| scale_function | a scale function (e.g. scale_fill_divergent) |

### Value

A function with the same arguments as scale_function that works with discretised values.

### See Also

scale_fill_discretised

### Examples

```
library(ggplot2)
scale_fill_brewer_discretised <- as.discretised_scale(scale_fill_distiller)
```

## as.path *Interpolates between locations*

### Description

This is a helper function to quickly make an interpolated list of locations between a number of locations

### Usage

```
as.path(x, y, n = 10, path = TRUE)
```

### Arguments

| | |
|---|---|
| x, y | numeric vectors of x and y locations. If one of them is of length 1, if will be recycled. |
| n | number of points to interpolate to |
| path | either TRUE of a character vector with the name of the path. |

### Details

This function is mostly useful when combined with Interpolate

### Value

A list of components x and y with the list of locations and the path arguments

### See Also

Interpolate

---

`ConvertLongitude` *Converts between longitude conventions*

---

### Description

Converts longitude from [0, 360) to [-180, 180) and vice versa.

### Usage

```
ConvertLongitude(lon, group = NULL, from = NULL)
```

### Arguments

| | |
|---|---|
| `lon` | numeric vector of longitude |
| `group` | optional vector of groups (the same length as longitude) that will be split on the edges (see examples) |
| `from` | optionally explicitly say from which convention to convert |

### Value

If `group` is missing, a numeric vector the same length of lon. Else, a list with vectors `lon` and `group`.

### Examples

```
library(ggplot2)
library(data.table)
data(geopotential)
ggplot(geopotential[date == date[1]], aes(lon, lat, z = gh)) +
    geom_contour(color = "black") +
    geom_contour(aes(x = ConvertLongitude(lon)))

map <- setDT(map_data("world"))
map[, c("lon", "group2") := ConvertLongitude(long, group, from = 180)]

ggplot(map, aes(lon, lat, group = group2)) +
    geom_path()
```

## coriolis *Effects of the Earth's rotation*

### Description

Coriolis and beta parameters by latitude.

### Usage

```
coriolis(lat)

f(lat)

coriolis.dy(lat, a = 6371000)

f.dy(lat, a = 6371000)
```

### Arguments

| | |
|---|---|
| lat | latitude in degrees |
| a | radius of the earth |

### Details

All functions use the correct sidereal day (24hs 56mins 4.091s) instead of the incorrect solar day (24hs) for 0.3\ pedantry.

## cut.eof *Remove some principal components.*

### Description

Returns an eof object with just the n principal components.

### Usage

```
## S3 method for class 'eof'
cut(x, n, ...)
```

### Arguments

| | |
|---|---|
| x | an eof object |
| n | which eofs to keep |
| ... | further arguments passed to or from other methods |

## denormalise *Denormalise eof matrices*

### Description

The matrices returned by `EOF()` are normalized. This function multiplies the left or right matrix by the diagonal matrix to return it to proper units.

### Usage

```
denormalise(eof, which = c("left", "right"))

denormalize(eof, which = c("left", "right"))
```

### Arguments

| | |
|---|---|
| `eof` | an `eof` object. |
| `which` | which side of the eof decomposition to denormalise |

## Derivate *Derivate a discrete variable using finite differences*

### Description

Derivate a discrete variable using finite differences

### Usage

```
Derivate(
  formula,
  order = 1,
  cyclical = FALSE,
  fill = FALSE,
  data = NULL,
  sphere = FALSE,
  a = 6371000,
  equispaced = TRUE
)

Laplacian(
  formula,
  cyclical = FALSE,
  fill = FALSE,
  data = NULL,
  sphere = FALSE,
```

```
  a = 6371000,
  equispaced = TRUE
)

Divergence(
  formula,
  cyclical = FALSE,
  fill = FALSE,
  data = NULL,
  sphere = FALSE,
  a = 6371000,
  equispaced = TRUE
)

Vorticity(
  formula,
  cyclical = FALSE,
  fill = FALSE,
  data = NULL,
  sphere = FALSE,
  a = 6371000,
  equispaced = TRUE
)
```

## Arguments

| | |
|---|---|
| formula | a formula indicating dependent and independent variables |
| order | order of the derivative |
| cyclical | logical vector of boundary condition for each independent variable |
| fill | logical indicating whether to fill values at the boundaries with forward and backwards differencing |
| data | optional data.frame containing the variables |
| sphere | logical indicating whether to use spherical coordinates (see details) |
| a | radius to use in spherical coordinates (defaults to Earth's radius) |
| equispaced | logical indicating whether points are equispaced or not. |

## Details

Each element of the return vector is an estimation of $\frac{\partial^n x}{\partial y^n}$ by centred finite differences.

If sphere = TRUE, then the first two independent variables are assumed to be longitude and latitude (**in that order**) in degrees. Then, a correction is applied to the derivative so that they are in the same units as a.

Using fill = TRUE will degrade the solution near the edges of a non-cyclical boundary. Use with caution.

Laplacian(), Divergence() and Vorticity() are convenient wrappers that call Derivate() and make the appropriate sums. For Divergence() and Vorticity(), formula must be of the form vx + vy ~ x + y (**in that order**).

## Value

If there is one independent variable and one dependent variable, a numeric vector of the same length as the dependent variable. If there are two or more independent variables or two or more dependent variables, a list containing the directional derivatives of each dependent variables.

## See Also

Other meteorology functions: `EOF()`, `GeostrophicWind()`, `WaveFlux()`, `thermodynamics`, `waves`

## Examples

```
theta <- seq(0, 360, length.out = 20)*pi/180
theta <- theta[-1]
x <- cos(theta)
dx_analytical <- -sin(theta)
dx_finitediff <- Derivate(x ~ theta, cyclical = TRUE)[[1]]

plot(theta, dx_analytical, type = "l")
points(theta, dx_finitediff, col = "red")

# Curvature (Laplacian)
# Note the different boundary conditions for each dimension
variable <- expand.grid(lon = seq(0, 360, by = 3)[-1],
                        lat = seq(-90, 90, by = 3))
variable$z <- with(variable, cos(lat*pi/180*3) + sin(lon*pi/180*2))
variable <- cbind(
     variable,
     as.data.frame(Derivate(z ~ lon + lat, data = variable,
                            cyclical = c(TRUE, FALSE), order = 2)))
library(ggplot2)
ggplot(variable, aes(lon, lat)) +
    geom_contour(aes(z = z)) +
    geom_contour(aes(z = z.ddlon + z.ddlat), color = "red")

# The same as
ggplot(variable, aes(lon, lat)) +
    geom_contour(aes(z = z)) +
    geom_contour(aes(z = Laplacian(z ~ lon + lat, cyclical = c(TRUE, FALSE))),
                 color = "red")
```

---

`DivideTimeseries` *Divides long timeseries for better reading*

---

## Description

Long timeseries can be compressed to the point of being unreadable when plotted on a page. This function takes a ggplot object of a timeseries and divides it into panels so that the time dimension gets stretched for better readability.

### Usage

```
DivideTimeseries(g, x, n = 2, xlab = "x", ylab = "y")
```

### Arguments

| | |
|---|---|
| g | ggplot object |
| x | The vector that was used in g for the x axis (must be of class Date) |
| n | Number of panels |
| xlab | x axis label |
| ylab | y axis label |

### Value

Draws a plot.

### See Also

Other ggplot2 helpers: MakeBreaks(), WrapCircular(), geom_arrow(), geom_contour2(), geom_contour_fill(), geom_label_contour(), geom_relief(), geom_streamline(), guide_colourstrip(), map_labels, reverselog_trans(), scale_divergent, scale_longitude, stat_na(), stat_subset()

### Examples

```
library(ggplot2)
library(data.table)
gdata <- geopotential[lat == -30 & lon == 0]
g <- ggplot(gdata, aes(date, gh)) +
    geom_line() +
    geom_smooth() +
    scale_x_date(date_breaks = "1 year", date_labels = "%b")
DivideTimeseries(g, gdata$date, n = 2, "Date", "Max Temperature")
```

---

## EOF *Empirical Orthogonal Function*

---

### Description

Computes Singular Value Decomposition (also known as Principal Components Analysis or Empirical Orthogonal Functions).

## Usage

```
EOF(
  formula,
  n = 1,
  data = NULL,
  B = 0,
  probs = c(lower = 0.025, mid = 0.5, upper = 0.975),
  rotate = FALSE,
  suffix = "PC",
  fill = NULL,
  engine = NULL
)
```

## Arguments

| | |
|---|---|
| formula | a formula to build the matrix that will be used in the SVD decomposition (see Details) |
| n | which singular values to return (if NULL, returns all) |
| data | a data.frame |
| B | number of bootstrap samples used to estimate confidence intervals. Ignored if <= 1. |
| probs | the probabilities of the lower and upper values of estimated confidence intervals. If named, it's names will be used as column names. |
| rotate | if TRUE, scores and loadings will be rotated using varimax |
| suffix | character to name the principal components |
| fill | value to infill implicit missing values or NULL if the data is dense. |
| engine | function to use to compute SVD. If NULL it uses irlba::irlba (if installed) if the largest singular value to compute is lower than half the maximum possible value, otherwise it uses base::svd. If the user provides a function, it needs to be a drop-in replacement for base::svd (the same arguments and output format). |

## Details

Singular values can be computed over matrices so `formula` denotes how to build a matrix from the data. It is a formula of the form VAR ~ LEFT | RIGHT (see Formula::Formula) in which VAR is the variable whose values will populate the matrix, and LEFT represent the variables used to make the rows and RIGHT, the columns of the matrix. Think it like "VAR *as a function* of LEFT *and* RIGHT". The variable combination used in this formula *must* identify an unique value in a cell.

So, for example, `v ~ x + y | t` would mean that there is one value of `v` for each combination of `x`, `y` and `t`, and that there will be one row for each combination of `x` and `y` and one row for each `t`.

In the result, the left and right vectors have dimensions of the LEFT and RIGHT part of the `formula`, respectively.

It is much faster to compute only some singular vectors, so is advisable not to set `n` to `NULL`. If the irlba package is installed, EOF uses irlba::irlba instead of base::svd since it's much faster.

The bootstrapping procedure follows Fisher et.al. (2016) and returns the standard deviation of each singular value.

## Value

An `eof` object which is just a named list of `data.tables`

**left** data.table with left singular vectors

**right** data.table with right singular vectors

**sdev** data.table with singular values, their explained variance, and, optionally, quantiles estimated via bootstrap

There are some methods implemented

- summary
- screeplot and the equivalent autoplot
- cut.eof
- predict

## References

Fisher, A., Caffo, B., Schwartz, B., & Zipunnikov, V. (2016). Fast, Exact Bootstrap Principal Component Analysis for p > 1 million. Journal of the American Statistical Association, 111(514), 846–860. doi: 10.1080/01621459.2015.1062383

## See Also

Other meteorology functions: `Derivate()`, `GeostrophicWind()`, `WaveFlux()`, `thermodynamics`, `waves`

## Examples

```
# The Antarctic Oscillation is computed from the
# monthly geopotential height anomalies weigthed by latitude.
library(data.table)
data(geopotential)
geopotential <- copy(geopotential)
geopotential[, gh.t.w := Anomaly(gh)*sqrt(cos(lat*pi/180)),
      by = .(lon, lat, month(date))]

eof <- EOF(gh.t.w ~ lat + lon | date, 1:5, data = geopotential,
           B = 100, probs = c(low = 0.1, hig = 0.9))

# Inspect the explained variance of each component
summary(eof)
screeplot(eof)

# Keep only the 1st.
aao <- cut(eof, 1)

# AAO field
library(ggplot2)
ggplot(aao$left, aes(lon, lat, z = gh.t.w)) +
    geom_contour(aes(color = ..level..)) +
```

```
    coord_polar()

# AAO signal
ggplot(aao$right, aes(date, gh.t.w)) +
    geom_line()

# standard deviation, % of explained variance and
# confidence intervals.
aao$sdev

# Reconstructed fields based only on the two first
# principal components
field <- predict(eof, 1:2)

# Compare it to the real field.
ggplot(field[date == date[1]], aes(lon, lat)) +
    geom_contour_fill(aes(z = gh.t.w), data = geopotential[date == date[1]]) +
    geom_contour2(aes(z = gh.t.w, linetype = factor(-sign(stat(level))))) +
    scale_fill_divergent()
```

---

EPflux *Computes Eliassen-Palm fluxes.*

---

## Description

Computes Eliassen-Palm fluxes.

## Usage

```
EPflux(lon, lat, lev, t, u, v)
```

## Arguments

| | |
|---|---|
| lon | longitudes in degrees. |
| lat | latitudes in degrees. |
| lev | pressure levels. |
| t | temperature in Kelvin. |
| u | zonal wind in m/s. |
| v | meridional wind in m/s. |

## Value

A data.table with columns Flon, Flat and Flev giving the zonal, meridional and vertical components of the EP Fluxes at each longitude, latitude and level.

## References

Plumb, R. A. (1985). On the Three-Dimensional Propagation of Stationary Waves. Journal of the Atmospheric Sciences, 42(3), 217–229. doi: 10.1175/15200469(1985)042<0217:OTTDPO>2.0.CO;2
Cohen, J., Barlow, M., Kushner, P. J., & Saito, K. (2007). Stratosphere–Troposphere Coupling and Links with Eurasian Land Surface Variability. Journal of Climate, 20(21), 5335–5343. doi: 10.1175/2007JCLI1725.1

---

`FitLm` *Fast estimates of linear regression*

---

## Description

Computes a linear regression with stats::.lm.fit and returns the estimate and, optionally, standard error for each regressor.

## Usage

```
FitLm(y, ..., weights = NULL, se = FALSE, r2 = se)

ResidLm(y, ..., weights = NULL)

Detrend(y, time = seq_along(y))
```

## Arguments

| | |
|---|---|
| `y` | numeric vector of observations to model |
| `...` | numeric vectors of variables used in the modelling |
| `weights` | numerical vector of weights (which doesn't need to be normalised) |
| `se` | logical indicating whether to compute the standard error |
| `r2` | logical indicating whether to compute r squared |
| `time` | time vector to use for detrending. Only necessary in the case of irregularly sampled timeseries |

## Value

FitLm returns a list with elements

**term** the name of the regressor

**estimate** estimate of the regression

**std.error** standard error

**df** degrees of freedom

**r.squared** Percent of variance explained by the model (repeated in each term)

**adj.r.squared** r.squared‘ adjusted based on the degrees of freedom)

ResidLm and Detrend returns a vector of the same length

If there's no complete cases in the regression, `NA`s are returned with no warning.

## Examples

```
# Linear trend with "signficant" areas shaded with points
library(data.table)
library(ggplot2)
system.time({
  regr <- geopotential[, FitLm(gh, date, se = TRUE), by = .(lon, lat)]
})

ggplot(regr[term != "(Intercept)"], aes(lon, lat)) +
    geom_contour(aes(z = estimate, color = ..level..)) +
    stat_subset(aes(subset = abs(estimate) > 2*std.error), size = 0.05)

# Using stats::lm() is much slower and with no names.
## Not run:
system.time({
  regr <- geopotential[, coef(lm(gh ~ date))[2], by = .(lon, lat)]
})

## End(Not run)
```

---

geom_arrow *Arrows*

---

## Description

Parametrization of ggplot2::geom_segment either by location and displacement or by magnitude and angle with default arrows. `geom_arrow()` is the same as `geom_vector()` but defaults to preserving the direction under coordinate transformation and different plot ratios.

## Usage

```
geom_arrow(
  mapping = NULL,
  data = NULL,
  stat = "arrow",
  position = "identity",
  ...,
  start = 0,
  direction = c("ccw", "cw"),
  pivot = 0.5,
  preserve.dir = TRUE,
  min.mag = 0,
  skip = 0,
  skip.x = skip,
  skip.y = skip,
  arrow.angle = 15,
  arrow.length = 0.5,
```

```
  arrow.ends = "last",
  arrow.type = "closed",
  arrow = grid::arrow(arrow.angle, grid::unit(arrow.length, "lines"), ends =
    arrow.ends, type = arrow.type),
  lineend = "butt",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

geom_vector(
  mapping = NULL,
  data = NULL,
  stat = "arrow",
  position = "identity",
  ...,
  start = 0,
  direction = c("ccw", "cw"),
  pivot = 0.5,
  preserve.dir = FALSE,
  min.mag = 0,
  skip = 0,
  skip.x = skip,
  skip.y = skip,
  arrow.angle = 15,
  arrow.length = 0.5,
  arrow.ends = "last",
  arrow.type = "closed",
  arrow = grid::arrow(arrow.angle, grid::unit(arrow.length, "lines"), ends =
    arrow.ends, type = arrow.type),
  lineend = "butt",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

## Arguments

| | |
|---|---|
| mapping | Set of aesthetic mappings created by aes() or aes_(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping. |
| data | The data to be displayed in this layer. There are three options:<br>If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot().<br>A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created.<br>A function will be called with a single argument, the plot data. The return |

| | |
|---|---|
| | value must be a `data.frame`, and will be used as the layer data. A `function` can be created from a `formula` (e.g. `~ head(.x,10)`). |
| `stat` | The statistical transformation to use on the data for this layer, as a string. |
| `position` | Position adjustment, either as a string, or the result of a call to a position adjustment function. |
| `...` | Other arguments passed on to `layer()`. These are often aesthetics, used to set an aesthetic to a fixed value, like `colour = "red"` or `size = 3`. They may also be parameters to the paired geom/stat. |
| `start` | starting angle for rotation in degrees |
| `direction` | direction of rotation (counter-clockwise or clockwise) |
| `pivot` | numeric indicating where to pivot the arrow where 0 means at the beginning and 1 means at the end. |
| `preserve.dir` | logical indicating whether to preserve direction or not |
| `min.mag` | minimum magnitude for plotting vectors |
| `skip, skip.x, skip.y` | numeric specifying number of gridpoints not to draw in the x and y direction |
| `arrow.length, arrow.angle, arrow.ends, arrow.type` | parameters passed to grid::arrow |
| `arrow` | specification for arrow heads, as created by arrow(). |
| `lineend` | Line end style (round, butt, square). |
| `na.rm` | If `FALSE`, the default, missing values are removed with a warning. If `TRUE`, missing values are silently removed. |
| `show.legend` | logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes. It can also be a named logical vector to finely select the aesthetics to display. |
| `inherit.aes` | If `FALSE`, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. `borders()`. |

## Details

Direction and start allows to work with different standards. For the meteorological standard, for example, use `star = -90` and `direction = "cw"`.

## Aesthetics

`geom_vector` understands the following aesthetics (required aesthetics are in bold)

- **x**
- **y**
- either **mag** and **angle**, or **dx** and **dy**
- `alpha`
- `colour`
- `linetype`
- `size`
- `lineend`

## See Also

Other ggplot2 helpers: DivideTimeseries(), MakeBreaks(), WrapCircular(), geom_contour2(), geom_contour_fill(), geom_label_contour(), geom_relief(), geom_streamline(), guide_colourstrip(), map_labels, reverselog_trans(), scale_divergent, scale_longitude, stat_na(), stat_subset()

## Examples

```
library(data.table)
library(ggplot2)

data(seals)
# If the velocity components are in the same units as the axis,
# geom_vector() (or geom_arrow(preserve.dir = TRUE)) might be a better option
ggplot(seals, aes(long, lat)) +
    geom_arrow(aes(dx = delta_long, dy = delta_lat), skip = 1, color = "red") +
    geom_vector(aes(dx = delta_long, dy = delta_lat), skip = 1) +
    scale_mag()

data(geopotential)
geopotential <- copy(geopotential)[date == date[1]]
geopotential[, gh.z := Anomaly(gh), by = .(lat)]
geopotential[, c("u", "v") := GeostrophicWind(gh.z, lon, lat)]

(g <- ggplot(geopotential, aes(lon, lat)) +
    geom_arrow(aes(dx = dlon(u, lat), dy = dlat(v)), skip.x = 3, skip.y = 2,
               color = "red") +
    geom_vector(aes(dx = dlon(u, lat), dy = dlat(v)), skip.x = 3, skip.y = 2) +
    scale_mag(max_size = 2, guide = "none"))

# A dramatic illustration of the difference between arrow and vector
g + coord_polar()

# When plotting winds in a lat-lon grid, a good way to have both
# the correct direction and an interpretable magnitude is to define
# the angle by the longitud and latitude displacement and the magnitude
# by the wind velocity. That way arrows are always parallel to streamlines
# and their magnitude are in the correct units.
ggplot(geopotential, aes(lon, lat)) +
    geom_contour(aes(z = gh.z)) +
    geom_vector(aes(angle = atan2(dlat(v), dlon(u, lat))*180/pi,
                    mag = Mag(v, u)), skip = 1, pivot = 0.5) +
    scale_mag()

# Sverdrup transport
library(data.table)
b <- 10
d <- 10
grid <- as.data.table(expand.grid(x = seq(1, d, by = 0.5),
                                  y = seq(1, b, by = 0.5)))
grid[, My := -sin(pi*y/b)*pi/b]
grid[, Mx := -pi^2/b^2*cos(pi*y/b)*(d - x)]
```

```
ggplot(grid, aes(x, y)) +
    geom_arrow(aes(dx = Mx, dy = My))

# Due to limitations in ggplot2 (see: https://github.com/tidyverse/ggplot2/issues/4291),
# if you define the vector with the dx and dy aesthetics, you need
# to explicitly add scale_mag() in order to show the arrow legend.

ggplot(grid, aes(x, y)) +
    geom_arrow(aes(dx = Mx, dy = My)) +
    scale_mag()

# Alternative, use Mag and Angle.
ggplot(grid, aes(x, y)) +
    geom_arrow(aes(mag = Mag(Mx, My), angle = Angle(Mx, My)))
```

---

`geom_contour2` *2d contours of a 3d surface*

---

## Description

Similar to ggplot2::geom_contour but it can label contour lines, accepts accepts a function as the `breaks` argument and and computes breaks globally instead of per panel.

## Usage

```
geom_contour2(
  mapping = NULL,
  data = NULL,
  stat = "contour2",
  position = "identity",
  ...,
  lineend = "butt",
  linejoin = "round",
  linemitre = 1,
  breaks = MakeBreaks(),
  bins = NULL,
  binwidth = NULL,
  global.breaks = TRUE,
  na.rm = FALSE,
  na.fill = FALSE,
  skip = 1,
  margin = grid::unit(c(1, 1, 1, 1), "pt"),
  label.placer = label_placer_flattest(),
  show.legend = NA,
  inherit.aes = TRUE
)
```

```
stat_contour2(
  mapping = NULL,
  data = NULL,
  geom = "contour2",
  position = "identity",
  ...,
  breaks = MakeBreaks(),
  bins = NULL,
  binwidth = NULL,
  kriging = FALSE,
  global.breaks = TRUE,
  na.rm = FALSE,
  na.fill = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

## Arguments

| | |
|---|---|
| mapping | Set of aesthetic mappings created by aes() or aes_(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping. |
| data | The data to be displayed in this layer. There are three options:<br>If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot().<br>A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created.<br>A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x,10)). |
| stat | The statistical transformation to use on the data for this layer, as a string. |
| position | Position adjustment, either as a string, or the result of a call to a position adjustment function. |
| ... | Other arguments passed on to layer(). These are often aesthetics, used to set an aesthetic to a fixed value, like colour = "red" or size = 3. They may also be parameters to the paired geom/stat. |
| lineend | Line end style (round, butt, square). |
| linejoin | Line join style (round, mitre, bevel). |
| linemitre | Line mitre limit (number greater than 1). |
| breaks | One of:<br>• A numeric vector of breaks<br>• A function that takes the range of the data and binwidth as input and returns breaks as output |
| bins | Number of evenly spaced breaks. |

| | |
|---|---|
| binwidth | Distance between breaks. |
| global.breaks | Logical indicating whether breaks should be computed for the whole data or for each grouping. |
| na.rm | If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed. |
| na.fill | How to fill missing values.<br>• FALSE for letting the computation fail with no interpolation<br>• TRUE for imputing missing values with Impute2D<br>• A numeric value for constant imputation<br>• A function that takes a vector and returns a numeric (e.g. mean) |
| skip | number of contours to skip for labelling (e.g. skip = 1 will skip 1 contour line between labels). |
| margin | the margin around labels around which contour lines are clipped to avoid overlapping. |
| label.placer | a label placer function. See label_placer_flattest(). |
| show.legend | logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display. |
| inherit.aes | If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders(). |
| geom | The geometric object to use display the data |
| kriging | Logical indicating whether to perform ordinary kriging before contouring. Use this if you want to use contours with irregularly spaced data. |

## Aesthetics

geom_contour2 understands the following aesthetics (required aesthetics are in bold):

Aesthetics related to contour lines:

- **x**
- **y**
- **z**
- alpha
- colour
- group
- linetype
- size
- weight

Aesthetics related to labels:

- label

- label_colour
- label_alpha
- label_size
- family
- fontface

## Computed variables

**level** height of contour

## See Also

Other ggplot2 helpers: DivideTimeseries(), MakeBreaks(), WrapCircular(), geom_arrow(), geom_contour_fill(), geom_label_contour(), geom_relief(), geom_streamline(), guide_colourstrip(), map_labels, reverselog_trans(), scale_divergent, scale_longitude, stat_na(), stat_subset()

Other ggplot2 helpers: DivideTimeseries(), MakeBreaks(), WrapCircular(), geom_arrow(), geom_contour_fill(), geom_label_contour(), geom_relief(), geom_streamline(), guide_colourstrip(), map_labels, reverselog_trans(), scale_divergent, scale_longitude, stat_na(), stat_subset()

## Examples

```
library(ggplot2)

# Breaks can be a function.
ggplot(reshape2::melt(volcano), aes(Var1, Var2)) +
    geom_contour2(aes(z = value, color = ..level..),
                  breaks = AnchorBreaks(130, binwidth = 10))

# Add labels by supplying the label aes.
ggplot(reshape2::melt(volcano), aes(Var1, Var2)) +
    geom_contour2(aes(z = value, label = ..level..))

ggplot(reshape2::melt(volcano), aes(Var1, Var2)) +
    geom_contour2(aes(z = value, label = ..level..),
                  skip = 0)

# Use label.placer to control where contours are labelled.
ggplot(reshape2::melt(volcano), aes(Var1, Var2)) +
    geom_contour2(aes(z = value, label = ..level..),
                      label.placer = label_placer_n(n = 2))

# Use the rot_adjuster argument of the placer function to
# control the angle. For example, to fix it to some angle:
ggplot(reshape2::melt(volcano), aes(Var1, Var2)) +
    geom_contour2(aes(z = value, label = ..level..),
                  skip = 0,
                  label.placer = label_placer_flattest(rot_adjuster = 0))
```

`geom_contour_fill` *Filled 2d contours of a 3d surface*

## Description

While ggplot2's `geom_contour` can plot nice contours, it doesn't work with the polygon geom. This stat makes some small manipulation of the data to ensure that all contours are closed and also computes a new aesthetic `int.level`, which differs from `level` (computed by ggplot2::geom_contour) in that represents the value of the `z` aesthetic *inside* the contour instead of at the edge. It also computes breaks globally instead of per panel, so that faceted plots have all the same binwidth.

## Usage

```
geom_contour_fill(
  mapping = NULL,
  data = NULL,
  stat = "ContourFill",
  position = "identity",
  ...,
  breaks = MakeBreaks(),
  bins = NULL,
  binwidth = NULL,
  kriging = FALSE,
  global.breaks = TRUE,
  na.fill = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

stat_contour_fill(
  mapping = NULL,
  data = NULL,
  geom = "polygon",
  position = "identity",
  ...,
  breaks = MakeBreaks(),
  bins = NULL,
  binwidth = NULL,
  global.breaks = TRUE,
  kriging = FALSE,
  na.fill = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

## Arguments

| | |
|---|---|
| mapping | Set of aesthetic mappings created by aes() or aes_(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping. |
| data | The data to be displayed in this layer. There are three options:<br>If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot().<br>A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created.<br>A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x,10)). |
| stat | The statistical transformation to use on the data for this layer, as a string. |
| position | Position adjustment, either as a string, or the result of a call to a position adjustment function. |
| ... | Other arguments passed on to layer(). These are often aesthetics, used to set an aesthetic to a fixed value, like colour = "red" or size = 3. They may also be parameters to the paired geom/stat. |
| breaks | numeric vector of breaks |
| bins | Number of evenly spaced breaks. |
| binwidth | Distance between breaks. |
| kriging | Logical indicating whether to perform ordinary kriging before contouring. Use this if you want to use contours with irregularly spaced data. |
| global.breaks | Logical indicating whether breaks should be computed for the whole data or for each grouping. |
| na.fill | How to fill missing values.<br>• FALSE for letting the computation fail with no interpolation<br>• TRUE for imputing missing values with Impute2D<br>• A numeric value for constant imputation<br>• A function that takes a vector and returns a numeric (e.g. mean) |
| show.legend | logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display. |
| inherit.aes | If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders(). |
| geom | The geometric object to use display the data |

## Aesthetics

geom_contour_fill understands the following aesthetics (required aesthetics are in bold):

- **x**
- **y**
- alpha
- colour
- group
- linetype
- size
- weight

### Computed variables

**level** An ordered factor that represents bin ranges.

**level_d** Same as `level`, but automatically uses `scale_fill_discretised()`

**level_low,level_high,level_mid** Lower and upper bin boundaries for each band, as well the mid point between the boundaries.

### See Also

Other ggplot2 helpers: `DivideTimeseries()`, `MakeBreaks()`, `WrapCircular()`, `geom_arrow()`, `geom_contour2()`, `geom_label_contour()`, `geom_relief()`, `geom_streamline()`, `guide_colourstrip()`, `map_labels`, `reverselog_trans()`, `scale_divergent`, `scale_longitude`, `stat_na()`, `stat_subset()`

### Examples

```
library(ggplot2)
surface <- reshape2::melt(volcano)
ggplot(surface, aes(Var1, Var2, z = value)) +
  geom_contour_fill() +
  geom_contour(color = "black", size = 0.1)

ggplot(surface, aes(Var1, Var2, z = value)) +
  geom_contour_fill(aes(fill = stat(level)))

ggplot(surface, aes(Var1, Var2, z = value)) +
  geom_contour_fill(aes(fill = stat(level_d)))
```

---

## geom_contour_tanaka *Illuminated contours*

---

### Description

Illuminated contours (aka Tanaka contours) use varying brightness and width to create an illusion of relief. This can help distinguishing between concave and convex areas (local minimums and maximums), specially in black and white plots or to make photocopy safe plots with divergent colour palettes, or to render a more aesthetically pleasing representation of topography.

## Usage

```
geom_contour_tanaka(
  mapping = NULL,
  data = NULL,
  stat = "Contour2",
  position = "identity",
  ...,
  breaks = NULL,
  bins = NULL,
  binwidth = NULL,
  sun.angle = 60,
  light = "white",
  dark = "gray20",
  range = c(0.01, 0.5),
  smooth = 0,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

## Arguments

| | |
|---|---|
| mapping | Set of aesthetic mappings created by aes() or aes_(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping. |
| data | The data to be displayed in this layer. There are three options:<br>If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot().<br>A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created.<br>A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x,10)). |
| stat | The statistical transformation to use on the data for this layer, as a string. |
| position | Position adjustment, either as a string, or the result of a call to a position adjustment function. |
| ... | Other arguments passed on to layer(). These are often aesthetics, used to set an aesthetic to a fixed value, like colour = "red" or size = 3. They may also be parameters to the paired geom/stat. |
| breaks | One of:<br>• A numeric vector of breaks<br>• A function that takes the range of the data and binwidth as input and returns breaks as output |
| bins | Number of evenly spaced breaks. |

| | |
|---|---|
| binwidth | Distance between breaks. |
| sun.angle | angle of the sun in degrees counterclockwise from 12 o' clock |
| light, dark | valid colour representing the light and dark shading |
| range | numeric vector of length 2 with the minimum and maximum size of lines |
| smooth | numeric indicating the degree of smoothing of illumination and size. Larger |
| na.rm | If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed. |
| show.legend | logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display. |
| inherit.aes | If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders(). |

## Aesthetics

geom_contour_tanaka understands the following aesthetics (required aesthetics are in bold)

- **x**
- **y**
- **z**
- linetype

## Examples

```
library(ggplot2)
library(data.table)
# A fresh look at the boring old volcano dataset
ggplot(reshape2::melt(volcano), aes(Var1, Var2)) +
    geom_contour_fill(aes(z = value)) +
    geom_contour_tanaka(aes(z = value)) +
    theme_void()

# If the transition between segments feels too abrupt,
# smooth it a bit with smooth
ggplot(reshape2::melt(volcano), aes(Var1, Var2)) +
    geom_contour_fill(aes(z = value)) +
    geom_contour_tanaka(aes(z = value), smooth = 1) +
    theme_void()

data(geopotential)
geo <- geopotential[date == unique(date)[4]]
geo[, gh.z := Anomaly(gh), by = lat]

# In a monochrome contour map, it's impossible to know which areas are
# local maximums or minimums.
ggplot(geo, aes(lon, lat)) +
    geom_contour2(aes(z = gh.z), color = "black", xwrap = c(0, 360))
```

```
# With tanaka contours, they are obvious.
ggplot(geo, aes(lon, lat)) +
    geom_contour_tanaka(aes(z = gh.z), dark = "black",
                        xwrap = c(0, 360)) +
    scale_fill_divergent()

# A good divergent color palette has the same luminosity for positive
# and negative values.But that means that printed in grayscale (Desaturated),
# they are indistinguishable.
(g <- ggplot(geo, aes(lon, lat)) +
    geom_contour_fill(aes(z = gh.z), xwrap = c(0, 360)) +
    scale_fill_gradientn(colours = c("#767676", "white", "#484848"),
                         values = c(0, 0.415, 1)))

# Tanaka contours can solve this issue.
g + geom_contour_tanaka(aes(z = gh.z))
```

---

`geom_label_contour` *Label contours*

---

## Description

Draws labels on contours built with ggplot2::stat_contour.

## Usage

```
geom_label_contour(
  mapping = NULL,
  data = NULL,
  stat = "text_contour",
  position = "identity",
  ...,
  min.size = 5,
  skip = 1,
  label.placer = label_placer_flattest(),
  parse = FALSE,
  nudge_x = 0,
  nudge_y = 0,
  label.padding = grid::unit(0.25, "lines"),
  label.r = grid::unit(0.15, "lines"),
  label.size = 0.25,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

```
geom_text_contour(
  mapping = NULL,
  data = NULL,
  stat = "text_contour",
  position = "identity",
  ...,
  min.size = 5,
  skip = 1,
  rotate = TRUE,
  label.placer = label_placer_flattest(),
  parse = FALSE,
  nudge_x = 0,
  nudge_y = 0,
  stroke = 0,
  check_overlap = FALSE,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

## Arguments

| | |
|---|---|
| mapping | Set of aesthetic mappings created by aes() or aes_(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping. |
| data | The data to be displayed in this layer. There are three options:<br>If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot().<br>A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created.<br>A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x,10)). |
| stat | The statistical transformation to use on the data for this layer, as a string. |
| position | Position adjustment, either as a string, or the result of a call to a position adjustment function. Cannot be jointy specified with nudge_x or nudge_y. |
| ... | Other arguments passed on to layer(). These are often aesthetics, used to set an aesthetic to a fixed value, like colour = "red" or size = 3. They may also be parameters to the paired geom/stat. |
| min.size | minimum number of points for a contour to be labelled. |
| skip | number of contours to skip |
| label.placer | a label placer function. See label_placer_flattest(). |
| parse | If TRUE, the labels will be parsed into expressions and displayed as described in ?plotmath. |

| | |
|---|---|
| nudge_x | Horizontal and vertical adjustment to nudge labels by. Useful for offsetting text from points, particularly on discrete scales. Cannot be jointly specified with position. |
| nudge_y | Horizontal and vertical adjustment to nudge labels by. Useful for offsetting text from points, particularly on discrete scales. Cannot be jointly specified with position. |
| label.padding | Amount of padding around label. Defaults to 0.25 lines. |
| label.r | Radius of rounded corners. Defaults to 0.15 lines. |
| label.size | Size of label border, in mm. |
| na.rm | If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed. |
| show.legend | logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display. |
| inherit.aes | If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders(). |
| rotate | logical indicating whether to rotate text following the contour. |
| stroke | numerical indicating width of stroke relative to the size of the text. Ignored if less than zero. |
| check_overlap | If TRUE, text that overlaps previous text in the same layer will not be plotted. check_overlap happens at draw time and in the order of the data. Therefore data should be arranged by the label column before calling geom_text(). Note that this argument is not supported by geom_label(). |

## Details

Is best used with a previous call to ggplot2::stat_contour with the same parameters (e.g. the same binwidth, breaks, or bins). Note that while geom_text_contour() can angle itself to follow the contour, this is not the case with geom_label_contour().

## Aesthetics

geom_text_contour understands the following aesthetics (required aesthetics are in bold):

- **x**
- **y**
- **label**
- alpha
- angle
- colour
- stroke.color
- family
- fontface

- group
- hjust
- lineheight
- size
- vjust

## See Also

Other ggplot2 helpers: DivideTimeseries(), MakeBreaks(), WrapCircular(), geom_arrow(), geom_contour2(), geom_contour_fill(), geom_relief(), geom_streamline(), guide_colourstrip(), map_labels, reverselog_trans(), scale_divergent, scale_longitude, stat_na(), stat_subset()

## Examples

```
library(ggplot2)
v <- reshape2::melt(volcano)
g <- ggplot(v, aes(Var1, Var2)) +
       geom_contour(aes(z = value))
g + geom_text_contour(aes(z = value))

g + geom_text_contour(aes(z = value), stroke = 0.2)

g + geom_text_contour(aes(z = value), stroke = 0.2, stroke.colour = "red")

g + geom_text_contour(aes(z = value, stroke.colour = ..level..), stroke = 0.2) +
    scale_colour_gradient(aesthetics = "stroke.colour", guide = "none")

g + geom_text_contour(aes(z = value), rotate = FALSE)

g + geom_text_contour(aes(z = value),
                      label.placer = label_placer_random())

g + geom_text_contour(aes(z = value),
                      label.placer = label_placer_n(3))

g + geom_text_contour(aes(z = value),
                      label.placer = label_placer_flattest())

g + geom_text_contour(aes(z = value),
                      label.placer = label_placer_flattest(ref_angle = 90))
```

---

# geom_relief *Relief Shading*

---

## Description

geom_relief() simulates shading caused by relief. Can be useful when plotting topographic data because relief shading might give a more intuitive impression of the shape of the terrain than contour lines or mapping height to colour. geom_shadow() projects shadows.

## Usage

```
geom_relief(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  sun.angle = 60,
  raster = TRUE,
  interpolate = TRUE,
  shadow = FALSE,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

geom_shadow(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  sun.angle = 60,
  range = c(0, 1),
  skip = 0,
  raster = TRUE,
  interpolate = TRUE,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

## Arguments

| | |
|---|---|
| mapping | Set of aesthetic mappings created by aes() or aes_(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping. |
| data | The data to be displayed in this layer. There are three options:<br>If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot().<br>A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created.<br>A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x,10)). |
| stat | The statistical transformation to use on the data for this layer, as a string. |

| | |
|---|---|
| position | Position adjustment, either as a string, or the result of a call to a position adjustment function. |
| ... | Other arguments passed on to layer(). These are often aesthetics, used to set an aesthetic to a fixed value, like colour = "red" or size = 3. They may also be parameters to the paired geom/stat. |
| sun.angle | angle from which the sun is shining, in degrees counterclockwise from 12 o' clock |
| raster | if TRUE (the default), uses ggplot2::geom_raster, if FALSE, uses ggplot2::geom_tile. |
| interpolate | If TRUE interpolate linearly, if FALSE (the default) don't interpolate. |
| shadow | if TRUE, adds also a layer of geom_shadow() |
| na.rm | If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed. |
| show.legend | logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display. |
| inherit.aes | If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders(). |
| range | transparency range for shadows |
| skip | data points to skip when casting shadows |

## Details

light and dark must be valid colours determining the light and dark shading (defaults to "white" and "gray20", respectively).

## Aesthetics

geom_relief() and geom_shadow() understands the following aesthetics (required aesthetics are in bold)

- **x**
- **y**
- **z**
- light
- dark
- sun.angle

## See Also

Other ggplot2 helpers: DivideTimeseries(), MakeBreaks(), WrapCircular(), geom_arrow(), geom_contour2(), geom_contour_fill(), geom_label_contour(), geom_streamline(), guide_colourstrip(), map_labels, reverselog_trans(), scale_divergent, scale_longitude, stat_na(), stat_subset()

## Examples

```
## Not run:
library(ggplot2)
ggplot(reshape2::melt(volcano), aes(Var1, Var2)) +
      geom_relief(aes(z = value))

## End(Not run)
```

---

`geom_streamline` *Streamlines*

---

## Description

Streamlines are paths that are always tangential to a vector field. In the case of a steady field, it's identical to the path of a massless particle that moves with the "flow".

## Usage

```
geom_streamline(
  mapping = NULL,
  data = NULL,
  stat = "streamline",
  position = "identity",
  ...,
  L = 5,
  min.L = 0,
  res = 1,
  S = NULL,
  dt = NULL,
  xwrap = NULL,
  ywrap = NULL,
  skip = 1,
  skip.x = skip,
  skip.y = skip,
  n = NULL,
  nx = n,
  ny = n,
  jitter = 1,
  jitter.x = jitter,
  jitter.y = jitter,
  arrow.angle = 6,
  arrow.length = 0.5,
  arrow.ends = "last",
  arrow.type = "closed",
  arrow = grid::arrow(arrow.angle, grid::unit(arrow.length, "lines"), ends =
    arrow.ends, type = arrow.type),
  lineend = "butt",
```

```
  na.rm = TRUE,
  show.legend = NA,
  inherit.aes = TRUE
)

stat_streamline(
  mapping = NULL,
  data = NULL,
  geom = "streamline",
  position = "identity",
  ...,
  L = 5,
  min.L = 0,
  res = 1,
  S = NULL,
  dt = NULL,
  xwrap = NULL,
  ywrap = NULL,
  skip = 1,
  skip.x = skip,
  skip.y = skip,
  n = NULL,
  nx = n,
  ny = n,
  jitter = 1,
  jitter.x = jitter,
  jitter.y = jitter,
  arrow.angle = 6,
  arrow.length = 0.5,
  arrow.ends = "last",
  arrow.type = "closed",
  arrow = grid::arrow(arrow.angle, grid::unit(arrow.length, "lines"), ends =
    arrow.ends, type = arrow.type),
  lineend = "butt",
  na.rm = TRUE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

## Arguments

| | |
|---|---|
| mapping | Set of aesthetic mappings created by aes() or aes_(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping. |
| data | The data to be displayed in this layer. There are three options:<br>If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot().<br>A data.frame, or other object, will override the plot data. All objects will be |

| | |
|---|---|
| | fortified to produce a data frame. See fortify() for which variables will be created.<br>A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x,10)). |
| stat | The statistical transformation to use on the data for this layer, as a string. |
| position | Position adjustment, either as a string, or the result of a call to a position adjustment function. |
| ... | Other arguments passed on to layer(). These are often aesthetics, used to set an aesthetic to a fixed value, like colour = "red" or size = 3. They may also be parameters to the paired geom/stat. |
| L, | typical length of a streamline in x and y units |
| min.L | minimum length of segments to show |
| res, | resolution parameter (higher numbers increases the resolution) |
| S | optional numeric number of timesteps for integration |
| dt | optional numeric size "timestep" for integration |
| xwrap, ywrap | vector of length two used to wrap the circular dimension. |
| skip | numeric specifying number of gridpoints not to draw in the x and y direction |
| skip.x | numeric specifying number of gridpoints not to draw in the x and y direction |
| skip.y | numeric specifying number of gridpoints not to draw in the x and y direction |
| n, nx, ny | optional numeric indicating the number of points to draw in the x and y direction (replaces skip if not NULL) |
| jitter, jitter.x, jitter.y | amount of jitter of the starting points |
| arrow.angle | parameters passed to grid::arrow |
| arrow.length | parameters passed to grid::arrow |
| arrow.ends | parameters passed to grid::arrow |
| arrow.type | parameters passed to grid::arrow |
| arrow | specification for arrow heads, as created by arrow(). |
| lineend | Line end style (round, butt, square). |
| na.rm | If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed. |
| show.legend | logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display. |
| inherit.aes | If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders(). |
| geom | The geometric object to use display the data |

## Details

Streamlines are computed by simple integration with a forward Euler method. By default, `stat_streamline()` computes `dt` and `S` from `L`, `res`, the resolution of the grid and the mean magnitude of the field. `S` is then defined as the number of steps necessary to make a streamline of length `L` under an uniform mean field and `dt` is chosen so that each step is no larger than the resolution of the data (divided by the `res` parameter). Be aware that this rule of thumb might fail in field with very skewed distribution of magnitudes.

Alternatively, `L` and/or `res` are ignored if `S` and/or `dt` are specified explicitly. This not only makes it possible to fine-tune the result but also divorces the integration parameters from the properties of the data and makes it possible to compare streamlines between different fields.

The starting grid is a semi regular grid defined, either by the resolution of the field and the `skip.x` and `skip.y` parameters o the `nx` and `ny` parameters, jittered by an amount proportional to the resolution of the data and the `jitter.x` and `jitter.y` parameters.

It might be important that the units of the vector field are compatible to the units of the x and y dimensions. For example, passing dx and dy in m/s on a longitude-latitude grid will might misleading results (see spherical).

Missing values are not permitted and the field must be defined on a regular grid, for now.

## Aesthetics

`stat_streamline` understands the following aesthetics (required aesthetics are in bold)

- **x**
- **y**
- **dx**
- **dy**
- `alpha`
- `colour`
- `linetype`
- `size`

## Computed variables

**step** step in the simulation

**dx** dx at each location of the streamline

**dy** dy at each location of the streamline

## See Also

Other ggplot2 helpers: `DivideTimeseries()`, `MakeBreaks()`, `WrapCircular()`, `geom_arrow()`, `geom_contour2()`, `geom_contour_fill()`, `geom_label_contour()`, `geom_relief()`, `guide_colourstrip()`, `map_labels`, `reverselog_trans()`, `scale_divergent`, `scale_longitude`, `stat_na()`, `stat_subset()`

## Examples

```
## Not run:
library(data.table)
library(ggplot2)
data(geopotential)

geopotential <- copy(geopotential)[date == date[1]]
geopotential[, gh.z := Anomaly(gh), by = .(lat)]
geopotential[, c("u", "v") := GeostrophicWind(gh.z, lon, lat)]

(g <- ggplot(geopotential, aes(lon, lat)) +
    geom_contour2(aes(z = gh.z), xwrap = c(0, 360)) +
    geom_streamline(aes(dx = dlon(u, lat), dy = dlat(v)), L = 60,
                    xwrap = c(0, 360)))

# The circular parameter is particularly important for polar coordinates
g + coord_polar()

# If u and v are not converted into degrees/second, the resulting
# streamlines have problems, specially near the pole.
ggplot(geopotential, aes(lon, lat)) +
    geom_contour(aes(z = gh.z)) +
    geom_streamline(aes(dx = u, dy = v), L = 50)

# The step variable can be mapped to size or alpha to
# get cute "drops". It's important to note that ..dx.. (the calculated variable)
# is NOT the same as dx (from the data).
ggplot(geopotential, aes(lon, lat)) +
    geom_streamline(aes(dx = dlon(u, lat), dy = dlat(v), alpha = ..step..,
                        color = sqrt(..dx..^2 + ..dy..^2), size = ..step..),
                        L = 40, xwrap = c(0, 360), res = 2, arrow = NULL,
                        lineend = "round") +
    scale_size(range = c(0, 0.6))

# Using topographic information to simulate "rivers" from slope
topo <- GetTopography(295, -55+360, -30, -42, res = 1/20)  # needs internet!
topo[, c("dx", "dy") := Derivate(h ~ lon + lat)]
topo[h <= 0, c("dx", "dy") := 0]

# See how in this example the integration step is too coarse in the
# western montanous region where the slope is much higher than in the
# flatlands of La Pampa at in the east.
ggplot(topo, aes(lon, lat)) +
    geom_relief(aes(z = h), interpolate = TRUE, data = topo[h >= 0]) +
    geom_contour(aes(z = h), breaks = 0, color = "black") +
    geom_streamline(aes(dx = -dx, dy = -dy), L = 10, skip = 3, arrow = NULL,
                    color = "#4658BD") +
    coord_quickmap()

## End(Not run)
```

## geopotential *Geopotential height*

### Description

Monthly geopotential field at 700hPa south of 20°S from January 1990 to December 2000.

### Usage

```
geopotential
```

### Format

A data.table with 53224 rows and 5 variables.

**lon** longitude in degrees

**lat** latitude in degrees

**lev** level in hPa

**gh** geopotential height in meters

**date** date

### Source

https://psl.noaa.gov/data/gridded/data.ncep.reanalysis.derived.pressure.html

## GeostrophicWind *Calculate geostrophic winds*

### Description

Geostrophic wind from a geopotential height field.

### Usage

```
GeostrophicWind(gh, lon, lat, cyclical = "guess", g = 9.81, a = 6371000)
```

### Arguments

| | |
|---|---|
| gh | geopotential height |
| lon | longitude in degrees |
| lat | latitude in degrees |
| cyclical | boundary condition for longitude (see details) |
| g | acceleration of gravity |
| a | Earth's radius |

## Details

If `cyclical = "guess"` (the default) the function will try to guess if `lon` covers the whole globe and set cyclical conditions accordingly. For more predictable results, set the boundary condition explicitly.

## Value

A named list with vectors for the zonal and meridional component of geostrophic wind.

## See Also

Other meteorology functions: `Derivate()`, `EOF()`, `WaveFlux()`, `thermodynamics`, `waves`

## Examples

```
data(geopotential)
geopotential <- data.table::copy(geopotential)
geopotential[date == date[1], c("u", "v") := GeostrophicWind(gh, lon, lat)]
library(ggplot2)
ggplot(geopotential[date == date[1]], aes(lon, lat)) +
    geom_contour(aes(z = gh)) +
    geom_vector(aes(dx = u, dy = v), skip = 2) +
    scale_mag()
```

---

# GetSMNData *Get Meteorological data*

---

## Description

Downloads minimum and maximum temperature station data from Argentina's National Weather Service's public access. Data availability is not guaranteed so you are encouraged to check it on the website.

## Usage

```
GetSMNData(
  date,
  type = c("hourly", "daily", "radiation"),
  bar = FALSE,
  cache = TRUE,
  file.dir = tempdir()
)
```

## Arguments

| | |
|---|---|
| `date` | date vector of dates to fetch data |
| `type` | type of data to retrieve |
| `bar` | logical object indicating whether to show a progress bar |
| `cache` | logical indicating if the results should be saved on disk |
| `file.dir` | optional directory where to save and/or retrieve data |

## Value

For `type = "hourly"`, a data.frame with observations of

**date** date

**t** temperature in degrees celsius

**rh** relative humidity in %

**slp** sea level pressure in hPa

**dir** wind direction in clockwise degrees from 6 o'clock

**V** wind magnitude in m/s

**station** station name

For `type = "daily"`, a data.frame with observations of

**date** date

**tmax** maximum daily temperature in degrees celsius

**tmin** minimum daily temperature in degrees celsius

**station** station name

For `type = "radiation"`, a data.frame with observations of

**date** date

**global** global radiation in W/m^2

**diffuse** diffuse radiation in W/m^2

**station** station name

## Source

https://ssl.smn.gob.ar/dpd/pron5d-calendario.php

## Examples

```
## Not run:
dates <- seq.Date(lubridate::today() - 30, lubridate::today(), by = "1 day")
data <- GetSMNData(dates, type = "daily", bar = TRUE)

library(ggplot2)
ggplot(subset(data, station == "BASE BELGRANO II"),
       aes(date, (tmax + tmin)/2)) +
```

```
    geom_line()

## End(Not run)
```

---

`GetTopography` *Get topographic data*

---

## Description

Retrieves topographic data from ETOPO1 Global Relief Model (see references).

## Usage

```
GetTopography(
  lon.west,
  lon.east,
  lat.north,
  lat.south,
  resolution = 3.5,
  cache = TRUE,
  file.dir = tempdir(),
  verbose = interactive()
)
```

## Arguments

| | |
|---|---|
| `lon.west, lon.east, lat.north, lat.south` | latitudes and longitudes of the bounding box in degrees |
| `resolution` | numeric vector indicating the desired resolution (in degrees) in the lon and lat directions (maximum resolution is 1 minute) |
| `cache` | logical indicating if the results should be saved on disk |
| `file.dir` | optional directory where to save and/or retrieve data |
| `verbose` | logical indicating whether to print progress |

## Details

Very large requests can take long and can be denied by the NOAA server. If the function fails, try with a smaller bounding box or coarser resolution.

Longitude coordinates must be between 0 and 360.

## Value

A data table with height (in meters) for each longitude and latitude.

## References

Source: Amante, C. and B.W. Eakins, 2009. ETOPO1 1 Arc-Minute Global Relief Model: Procedures, Data Sources and Analysis. NOAA Technical Memorandum NESDIS NGDC-24. National Geophysical Data Center, NOAA. doi: 10.7289/V5C8276M

## Examples

```
## Not run:
topo <- GetTopography(280, 330, 0, -60, resolution = 0.5)
library(ggplot2)
ggplot(topo, aes(lon, lat)) +
    geom_raster(aes(fill = h)) +
    geom_contour(aes(z = h), breaks = 0, color = "black", size = 0.3) +
    scale_fill_gradient2(low = "steelblue", high = "goldenrod2", mid = "olivedrab") +
    coord_quickmap()

## End(Not run)
```

---

# guide_vector *Reference arrow for magnitude scales*

---

## Description

Draws a reference arrow. Highly experimental.

## Usage

```
guide_vector(
  title = ggplot2::waiver(),
  title.position = NULL,
  title.theme = NULL,
  title.hjust = NULL,
  title.vjust = NULL,
  label = TRUE,
  label.position = NULL,
  label.theme = NULL,
  label.hjust = NULL,
  label.vjust = NULL,
  keywidth = NULL,
  keyheight = NULL,
  direction = NULL,
  default.unit = "cm",
  override.aes = list(),
  nrow = NULL,
  ncol = NULL,
  byrow = FALSE,
  reverse = FALSE,
```

```
  order = 0,
  ...
)
```

## Arguments

| | |
|---|---|
| title | A character string or expression indicating a title of guide. If NULL, the title is not shown. By default (waiver()), the name of the scale object or the name specified in labs() is used for the title. |
| title.position | A character string indicating the position of a title. One of "top" (default for a vertical guide), "bottom", "left" (default for a horizontal guide), or "right." |
| title.theme | A theme object for rendering the title text. Usually the object of element_text() is expected. By default, the theme is specified by legend.title in theme() or theme. |
| title.hjust | A number specifying horizontal justification of the title text. |
| title.vjust | A number specifying vertical justification of the title text. |
| label | logical. If TRUE then the labels are drawn. If FALSE then the labels are invisible. |
| label.position | A character string indicating the position of a label. One of "top", "bottom" (default for horizontal guide), "left", or "right" (default for vertical guide). |
| label.theme | A theme object for rendering the label text. Usually the object of element_text() is expected. By default, the theme is specified by legend.text in theme(). |
| label.hjust | A numeric specifying horizontal justification of the label text. |
| label.vjust | A numeric specifying vertical justification of the label text. |
| keywidth | A numeric or a grid::unit() object specifying the width of the legend key. Default value is legend.key.width or legend.key.size in theme(). |
| keyheight | A numeric or a grid::unit() object specifying the height of the legend key. Default value is legend.key.height or legend.key.size in theme(). |
| direction | A character string indicating the direction of the guide. One of "horizontal" or "vertical." |
| default.unit | A character string indicating grid::unit() for keywidth and keyheight. |
| override.aes | A list specifying aesthetic parameters of legend key. See details and examples. |
| nrow | The desired number of rows of legends. |
| ncol | The desired number of column of legends. |
| byrow | logical. If FALSE (the default) the legend-matrix is filled by columns, otherwise the legend-matrix is filled by rows. |
| reverse | logical. If TRUE the order of legends is reversed. |
| order | positive integer less than 99 that specifies the order of this guide among multiple guides. This controls the order in which multiple guides are displayed, not the contents of the guide itself. If 0 (default), the order is determined by a secret algorithm. |
| ... | ignored. |

## See Also

scale_vector

## Impute2D *Impute missing values by linear or constant interpolation*

### Description

Provides methods for (soft) imputation of missing values.

### Usage

```
Impute2D(formula, data = NULL, method = "interpolate")
```

### Arguments

| | |
|---|---|
| formula | a formula indicating dependent and independent variables (see Details) |
| data | optional data.frame with the data |
| method | "interpolate" for interpolation, a numeric for constant imputation or a function that takes a vector and returns a number (like mean) |

### Details

This is "soft" imputation because the imputed values are not supposed to be representative of the missing data but just filling for algorithms that need complete data (in particular, contouring). The method used if `method = "interpolate"` is to do simple linear interpolation in both the x and y direction and then average the result.

This is the imputation method used by `geom_contour_fill()`.

## ImputeEOF *Impute missing values*

### Description

Imputes missing values via Data Interpolating Empirical Orthogonal Functions (DINEOF).

### Usage

```
ImputeEOF(
  formula,
  max.eof = NULL,
  data = NULL,
  min.eof = 1,
  tol = 0.01,
  max.iter = 10000,
  validation = NULL,
  verbose = interactive()
)
```

## Arguments

| | |
|---|---|
| `formula` | a formula to build the matrix that will be used in the SVD decomposition (see Details) |
| `max.eof, min.eof` | maximum and minimum number of singular values used for imputation |
| `data` | a data.frame |
| `tol` | tolerance used for determining convergence |
| `max.iter` | maximum iterations allowed for the algorithm |
| `validation` | number of points to use in cross-validation (defaults to the maximum of 30 or 10% of the non NA points) |
| `verbose` | logical indicating whether to print progress |

## Details

Singular values can be computed over matrices so `formula` denotes how to build a matrix from the data. It is a formula of the form VAR ~ LEFT | RIGHT (see Formula::Formula) in which VAR is the variable whose values will populate the matrix, and LEFT represent the variables used to make the rows and RIGHT, the columns of the matrix. Think it like "VAR *as a function* of LEFT *and* RIGHT".

Alternatively, if `value.var` is not `NULL`, it's possible to use the (probably) more familiar data.table::dcast formula interface. In that case, `data` must be provided.

If `data` is a matrix, the `formula` argument is ignored and the function returns a matrix.

## Value

A vector of imputed values with attributes `eof`, which is the number of singular values used in the final imputation; and `rmse`, which is the Root Mean Square Error estimated from cross-validation.

## References

Beckers, J.-M., Barth, A., and Alvera-Azcárate, A.: DINEOF reconstruction of clouded images including error maps – application to the Sea-Surface Temperature around Corsican Island, Ocean Sci., 2, 183-199, doi: 10.5194/os21832006, 2006.

## Examples

```
library(data.table)
data(geopotential)
geopotential <- copy(geopotential)
geopotential[, gh.t := Anomaly(gh), by = .(lat, lon, month(date))]

# Add gaps to field
geopotential[, gh.gap := gh.t]
set.seed(42)
geopotential[sample(1:.N, .N*0.3), gh.gap := NA]

max.eof <- 5    # change to a higher value
```

```
geopotential[, gh.impute := ImputeEOF(gh.gap ~ lat + lon | date, max.eof,
                                      verbose = TRUE, max.iter = 2000)]

library(ggplot2)
ggplot(geopotential[date == date[1]], aes(lon, lat)) +
    geom_contour(aes(z = gh.t), color = "black") +
    geom_contour(aes(z = gh.impute))

# Scatterplot with a sample.
na.sample <- geopotential[is.na(gh.gap)][sample(1:.N, .N*0.1)]
ggplot(na.sample, aes(gh.t, gh.impute)) +
    geom_point()

# Estimated RMSE
attr(geopotential$gh.impute, "rmse")
# Real RMSE
geopotential[is.na(gh.gap), sqrt(mean((gh.t - gh.impute)^2))]
```

---

`Interpolate` *Bilinear interpolation*

---

## Description

Uses fields::interp.surface to interpolate values defined in a bidimensional grid with bilinear interpolation.

## Usage

```
Interpolate(formula, x.out, y.out, data = NULL, grid = TRUE, path = FALSE)
```

## Arguments

| | |
|---|---|
| `formula` | a formula indicating dependent and independent variables (see Details) |
| `x.out, y.out` | x and y values where to interpolate (see Details) |
| `data` | optional data.frame with the data |
| `grid` | logical indicating if x.out and y.out define a regular grid. |
| `path` | a logical or character indicating if the x.out and y.out define a path. If character, it will be the name of the column returning the order of said path. |

## Details

`formula` must be of the form VAR1 | VAR2 ~ X + Y where VAR1, VAR2, etc... are the names of the variables to interpolate and X and Y the names of the x and y values, respectively. It is also possible to pass only values of x, in which case, regular linear interpolation is performed and y.out, if exists, is ignored with a warning.

If `grid = TRUE`, `x.out` and `y.out` must define the values of a regular grid. If `grid = FALSE`, they define the locations where to interpolate. Both `grid` and `path` cannot be set to `TRUE` and the value of `path` takes precedence.

`x.out` can be a list, in which case, the first two elements will be interpreted as the x and y values where to interpolate and it can also have a `path` element that will be used in place of the `path` argument. This helps when creating a path with as.path (see Examples)

## Value

A data.frame with interpolated values and locations

## Examples

```
library(data.table)
data(geopotential)
geopotential <- geopotential[date == date[1]]
# new grid
x.out <- seq(0, 360, by = 10)
y.out <- seq(-90, 0, by = 10)

# Interpolate values to a new grid
interpolated <- geopotential[, Interpolate(gh ~ lon + lat, x.out, y.out)]

# Add values to an existing grid
geopotential[, gh.new := Interpolate(gh ~ lon + lat, lon, lat,
                                     data = interpolated, grid = FALSE)$gh]

# Interpolate multiple values
geopotential[, c("u", "v") := GeostrophicWind(gh, lon, lat)]
interpolated <- geopotential[, Interpolate(u | v ~ lon + lat, x.out, y.out)]

# Interpolate values following a path
lats <- c(-34, -54, -30)   # start and end latitudes
lons <- c(302, 290, 180)   # start and end longituded
path <- geopotential[, Interpolate(gh ~ lon + lat, as.path(lons, lats))]
```

---

# is.cross *Cross pattern*

---

## Description

Reduces the density of a regular grid using a cross pattern.

## Usage

```
is.cross(x, y, skip = 0)

cross(x, y)
```

## Arguments

| | |
|---|---|
| x, y | x and y points that define a regular grid. |
| skip | how many points to skip. Greater value reduces the final point density. |

## Value

`is.cross` returns a logical vector indicating whether each point belongs to the reduced grid or not. `cross` returns a list of x and y components of the reduced density grid.

## Examples

```
# Basic usage
grid <- expand.grid(x = 1:10, y = 1:10)
cross <- is.cross(grid$x, grid$y, skip = 2)

with(grid, plot(x, y))
with(grid, points(x[cross], y[cross], col = "red"))

# Its intended use is to highlight areas with geom_subset()
# with reduced densnity. This "hatches" areas with temperature
# over 270K
library(ggplot2)
ggplot(temperature[lev == 500], aes(lon, lat)) +
  geom_raster(aes(fill = air)) +
  stat_subset(aes(subset = air > 270 & is.cross(lon, lat)),
              geom = "point", size = 0.1)
```

---

# JumpBy *Skip observations*

---

## Description

Skip observations

## Usage

```
JumpBy(x, by, start = 1, fill = NULL)
```

## Arguments

| | |
|---|---|
| x | vector |
| by | numeric interval between elements to keep |
| start | index to start from |
| fill | how observations are skipped |

### Details

Mostly useful for labelling only every byth element.

### Value

A vector of the same class as x and, if `fill` is not `null`, the same length.

### See Also

Other utilities: `Anomaly()`, `Mag()`, `Percentile()`, `logic`

### Examples

```
x <- 1:50
JumpBy(x, 2)   # only odd numbers
JumpBy(x, 2, start = 2)   # only even numbers
JumpBy(x, 2, fill = NA)   # even numbers replaced by NA
JumpBy(x, 2, fill = 6)   # even numbers replaced by 6
```

---

## `logic` *Extended logical operators*

---

### Description

Extended binary operators for easy subsetting.

### Usage

```
x %~% target

Similar(x, target, tol = Inf)
```

### Arguments

| | |
|---|---|
| `x, target` | numeric vectors |
| `tol` | tolerance for similarity |

### Details

`%~%` can be thought as a "similar" operator. It's a fuzzy version of `%in%` in that returns `TRUE` for the element of `x` which is the (first) closest to any element of `target`.

`Similar` is a functional version of `%~%` that also has a `tol` parameter that indicates the maximum allowed tolerance.

### Value

A logical vector of the same length of x.

## See Also

Other utilities: `Anomaly()`, `JumpBy()`, `Mag()`, `Percentile()`

## Examples

```
set.seed(198)
x <- rnorm(100)
x[x %~% c(0.3, 0.5, 1)]

# Practical use case: vertical cross-section at
# approximately 36W between 50S and 50N.
cross.lon <- -34 + 360
library(ggplot2)
library(data.table)
ggplot(temperature[lon %~% cross.lon & lat %between% c(-50, 50)],
       aes(lat, lev)) +
    geom_contour(aes(z = air))
```

---

`Mag` *Magnitude and angle of a vector*

---

## Description

Computes the magnitude of a vector of any dimension. Or angle (in degrees) in 2 dimensions.

## Usage

```
Mag(...)

Angle(x, y)
```

## Arguments

| | |
|---|---|
| `...` | numeric vectors of coordinates or list of coordinates |
| `x, y,` | x and y directions of the vector |

## Details

Helpful to save keystrokes and gain readability when computing wind (or any other vector quantity) magnitude.

## Value

`Mag`: A numeric vector the same length as each element of ... that is $\sqrt{(x^2 + y^2 + ...)}$. `Angle`: A numeric vector of the same length as x and y that is `atan2(y,x)*180/pi`.

## See Also

Other utilities: Anomaly(), JumpBy(), Percentile(), logic

Other utilities: Anomaly(), JumpBy(), Percentile(), logic

## Examples

```
Mag(10, 10)
Angle(10, 10)
Mag(10, 10, 10, 10)
Mag(list(10, 10, 10, 10))

# There's no vector recicling!
## Not run:
Mag(1, 1:2)

## End(Not run)
```

---

MakeBreaks *Functions for making breaks*

---

## Description

Functions that return functions suitable to use as the breaks argument in ggplot2's continuous scales and in geom_contour_fill.

## Usage

```
MakeBreaks(binwidth = NULL, bins = 10, exclude = NULL)

AnchorBreaks(anchor = 0, binwidth = NULL, exclude = NULL, bins = 10)
```

## Arguments

| | |
|---|---|
| binwidth | width of breaks |
| bins | number of bins, used if binwidth = NULL |
| exclude | a vector of breaks to exclude |
| anchor | anchor value |

## Details

MakeBreaks is essentially an export of the default way ggplot2::stat_contour makes breaks.

AnchorBreaks makes breaks starting from an anchor value and covering the range of the data according to binwidth.

## Value

A function that takes a range as argument and a binwidth as an optional argument and returns a sequence of equally spaced intervals covering the range.

## See Also

Other ggplot2 helpers: DivideTimeseries(), WrapCircular(), geom_arrow(), geom_contour2(), geom_contour_fill(), geom_label_contour(), geom_relief(), geom_streamline(), guide_colourstrip(), map_labels, reverselog_trans(), scale_divergent, scale_longitude, stat_na(), stat_subset()

## Examples

```
my_breaks <- MakeBreaks(10)
my_breaks(c(1, 100))
my_breaks(c(1, 100), 20)    # optional new binwidth argument ignored

MakeBreaks()(c(1, 100), 20)  # but is not ignored if initial binwidth is NULL

# One to one mapping between contours and breaks
library(ggplot2)
binwidth <- 20
ggplot(reshape2::melt(volcano), aes(Var1, Var2, z = value)) +
    geom_contour(aes(color = ..level..), binwidth = binwidth) +
    scale_color_continuous(breaks = MakeBreaks(binwidth))

#Two ways of getting the same contours. Better use the second one.
ggplot(reshape2::melt(volcano), aes(Var1, Var2, z = value)) +
    geom_contour2(aes(color = ..level..), breaks = AnchorBreaks(132),
                  binwidth = binwidth) +
    geom_contour2(aes(color = ..level..), breaks = AnchorBreaks(132, binwidth)) +
    scale_color_continuous(breaks = AnchorBreaks(132, binwidth))
```

---

# map_labels *Label longitude and latitude*

---

## Description

Provide easy functions for adding suffixes to longitude and latitude for labelling maps.

## Usage

```
LonLabel(lon, east = "°E", west = "°W", zero = "°")

LatLabel(lat, north = "°N", south = "°S", zero = "°")
```

### Arguments

| | |
|---|---|
| `lon` | longitude in degrees |
| `east, west, north, south, zero` | text to append for each quadrant |
| `lat` | latitude in degrees |

### Details

The default values are for Spanish.

### See Also

Other ggplot2 helpers: `DivideTimeseries()`, `MakeBreaks()`, `WrapCircular()`, `geom_arrow()`, `geom_contour2()`, `geom_contour_fill()`, `geom_label_contour()`, `geom_relief()`, `geom_streamline()`, `guide_colourstrip()`, `reverselog_trans()`, `scale_divergent`, `scale_longitude`, `stat_na()`, `stat_subset()`

### Examples

```
LonLabel(0:360)
```

---

## MaskLand *Mask*

---

### Description

Creates a mask

### Usage

```
MaskLand(lon, lat, mask = "world", wrap = c(0, 360))
```

### Arguments

| | |
|---|---|
| `lon` | a vector of longitudes in degrees in 0-360 format |
| `lat` | a vector of latitudes in degrees |
| `mask` | the name of the dataset (that will be load with `map`) for creating the mask |
| `wrap` | the longitude range to be used for a global mask |

### Value

A logical vector of the same length as lat and lon where `TRUE` means that the point is inside one of the polygons making up the map. For a global map (the default), this means that the point is over land.

## Examples

```
# Make a sea-land mask
mask <- temperature[lev == 1000, .(lon = lon, lat = lat, land = MaskLand(lon, lat))]
temperature <- temperature[mask, on = c("lon", "lat")]

# Take the temperature difference between land and ocean
diftemp <- temperature[,
          .(tempdif = mean(air[land == TRUE]) - mean(air[land == FALSE])),
           by = .(lat, lev)]
library(ggplot2)
ggplot(diftemp, aes(lat, lev)) +
    geom_contour(aes(z = tempdif, color = ..level..)) +
    scale_y_level() +
    scale_x_latitude() +
    scale_color_divergent()

# Mean temperature in the USA
usatemp <- temperature[, usa := MaskLand(lon, lat, mask = "usa")][
    , .(air = weighted.mean(air, cos(lat*pi/180))), by = .(usa, lev)][
        usa == TRUE]

ggplot(usatemp, aes(lev, air)) +
    geom_line() +
    scale_x_level() +
    coord_flip()
```

---

metR *metR: Tools for Easier Analysis of Meteorological Fields*

---

## Description

Many useful functions and extensions for dealing with meteorological data in the tidy data framework. Extends 'ggplot2' for better plotting of scalar and vector fields and provides commonly used analysis methods in the atmospheric sciences.

## Overview

Conceptually it's divided into *visualization tools* and *data tools*. The former are geoms, stats and scales that help with plotting using ggplot2, such as stat_contour_fill or scale_y_level, while the later are functions for common data processing tools in the atmospheric sciences, such as Derivate or EOF; these are implemented to work in the data.table paradigm, but also work with regular data frames.

To get started, check the vignettes:

- Visualization Tools: `vignette("Visualization-tools",package = "metR")`
- Working with Data: `vignette("Working-with-data",package = "metR")`

## Author(s)

**Maintainer**: Elio Campitelli <elio.campitelli@cima.fcen.uba.ar> (ORCID)

## See Also

Useful links:

- https://github.com/eliocamp/metR
- Report bugs at https://github.com/eliocamp/metR/issues

---

`Percentile` *Percentiles*

---

## Description

Computes percentiles.

## Usage

```
Percentile(x)
```

## Arguments

| | |
|---|---|
| x | numeric vector |

## Value

A numeric vector of the same length as x with the percentile of each value of x.

## See Also

Other utilities: `Anomaly()`, `JumpBy()`, `Mag()`, `logic`

## Examples

```
x <- rnorm(100)
p <- Percentile(x)
```

ReadNetCDF *Read NetCDF files.*

## Description

Using the ncdf4-package package, it reads a NetCDF file. The advantage over using ncvar_get is that the output is a tidy data.table with proper dimensions.

## Usage

```
ReadNetCDF(
  file,
  vars = NULL,
  out = c("data.frame", "vector", "array"),
  subset = NULL,
  key = FALSE
)

GlanceNetCDF(file, ...)
```

## Arguments

| | |
|---|---|
| file | source to read from. Must be one of:<br>• A string representing a local file with read access.<br>• A string representing a URL readable by ncdf4::nc_open(). (this includes DAP urls).<br>• A netcdf object returned by ncdf4::nc_open(). |
| vars | one of:<br>• NULL: reads all variables.<br>• a character vector with the name of the variables to read.<br>• a function that takes a vector with all the variables and returns either a character vector with the name of variables to read or a numeric/logical vector that indicates a subset of variables. |
| out | character indicating the type of output desired |
| subset | a list of subsetting objects. See below. |
| key | if TRUE, returns a data.table keyed by the dimensions of the data. |
| ... | in GlanceNetCDF(), ignored. Is there for convenience so that a call to ReadNetCDF() can be also valid for GlanceNetCDF(). |

## Value

The return format is specified by out. It can be a data table in which each column is a variable and each row, an observation; an array with named dimensions; or a vector. Since it's possible to return multiple arrays or vectors (one for each variable), for consistency the return type is always a list.

Either of these two options are much faster than the first since the most time consuming part is the melting of the array returned by ncdf4::ncvar_get. `out = "vector"` is particularly useful for adding new variables to an existing data frame with the same dimensions.

When not all variables specified in `vars` have the same number of dimensions, the shorter variables will be recycled. E.g. if reading a 3D pressure field and a 2D surface temperature field, the latter will be turned into a 3D field with the same values in each missing dimension.

`GlanceNetCDF()` returns a list of variables and dimensions included in the file with a nice printing method.

## Subsetting

In the most basic form, `subset` will be a named list whose names must match the dimensions specified in the NetCDF file and each element must be a vector whose range defines a contiguous subset of data. You don't need to provide and exact range that matches the actual gridpoints of the file; the closest gridpoint will be selected. Furthermore, you can use `NA` to refer to the existing minimum or maximum.

So, if you want to get Southern Hemisphere data from the from a file that defines latitude as `lat`, then you can use:

```
subset = list(lat = -90:0)
```

More complex subsetting operations are supported. If you want to read non-contiguous chunks of data, you can specify each chunk into a list inside `subset`. For example this subset

```
subset = list(list(lat = -90:-70, lon = 0:60),
              list(lat = 70:90, lon = 300:360))
```

will return two contiguous chunks: one on the South-West corner and one on the North-East corner. Alternatively, if you want to get the four corners that are combination of those two conditions,

```
subset = list(lat = list(-90:-70, 70:90),
              lon = list(0:60, 300:360))
```

Both operations can be mixed together. So for example this

```
subset = list(list(lat = -90:-70,
                   lon = 0:60),
              time = list(c("2000-01-01", "2000-12-31"),
                          c("2010-01-01", "2010-12-31")))
```

returns one spatial chunk for each of two temporal chunks.

The general idea is that named elements define 'global' subsets ranges that will be applied to every other subset, while each unnamed element define one contiguous chunk. In the above example, `time` defines two temporal ranges that every subset of data will have.

The above example, then, is equivalent to

```
subset = list(list(lat = -90:-70,
                   lon = 0:60,
                   time = c("2000-01-01", "2000-12-31")),
              list(lat = -90:-70,
                   lon = 0:60,
                   time = c("2010-01-01", "2010-12-31")))
```

but demands much less typing.

## Examples

```
file <- system.file("extdata", "temperature.nc", package = "metR")
# Get a list of variables.
variables <- GlanceNetCDF(file)
print(variables)

# The object returned by GlanceNetCDF is a list with lots
# of information
str(variables)

# Read only the first one, with name "var".
field <- ReadNetCDF(file, vars = c(var = names(variables$vars[1])))
# Add a new variable.
# ¡Make sure it's on the same exact grid!
field[, var2 := ReadNetCDF(file, out = "vector")]

## Not run:
# Using a DAP url
url <- "http://iridl.ldeo.columbia.edu/SOURCES/.Models/.SubX/.GMAO/.GEOS_V2p1/.hindcast/.ua/dods"
field <- ReadNetCDF(url, subset = list(M = 1,
                                       P = 10,
                                       S = "1999-01-01"))

# In this case, opening the netcdf file takes a non-neglible
# amount of time. So if you want to iterate over many dimensions,
# then it's more efficient to open the file first and then read it.

ncfile <- ncdf4::nc_open(url)
field <- ReadNetCDF(ncfile, subset = list(M = 1,
                                       P = 10,
                                       S = "1999-01-01"))


# Using a function in `vars` to read all variables that
# start with "radar_".
ReadNetCDF(radar_file, vars = \(x) startsWith(x, "radar_"))


## End(Not run)
```

## reverselog_trans *Reverse log transform*

### Description

Reverse log transformation. Useful when plotting and one axis is in pressure levels.

### Usage

```
reverselog_trans(base = 10)
```

### Arguments

| | |
|---|---|
| base | Base of the logarithm |

### See Also

Other ggplot2 helpers: DivideTimeseries(), MakeBreaks(), WrapCircular(), geom_arrow(), geom_contour2(), geom_contour_fill(), geom_label_contour(), geom_relief(), geom_streamline(), guide_colourstrip(), map_labels, scale_divergent, scale_longitude, stat_na(), stat_subset()

### Examples

```
# Adiabatic temperature profile
gamma <- 0.286
t <- data.frame(p = c(1000, 950, 850, 700, 500, 300, 200, 100))
t$t <- 300*(t$p/1000)^gamma

library(ggplot2)
ggplot(t, aes(p, t)) +
   geom_line() +
   coord_flip() +
   scale_x_continuous(trans = "reverselog")
```

## sa_pressure *Standard atmosphere*

### Description

Utilities to use the International Standard Atmosphere. It uses the International Standard Atmosphere up to the tropopause (11 km by definition) and then extends up to the 500 km using the ARDC Model Atmosphere.

## Usage

```
sa_pressure(height)

sa_height(pressure)

sa_temperature(height)

sa_height_trans(pressure_in = "hPa", height_in = "km")

sa_pressure_trans(height_in = "km", pressure_in = "hPa")

sa_height_breaks(n = 6, pressure_in = "hPa", height_in = "km", ...)

sa_height_axis(
  name = ggplot2::waiver(),
  breaks = sa_height_breaks(pressure_in = pressure_in, height_in = height_in),
  labels = ggplot2::waiver(),
  guide = ggplot2::waiver(),
  pressure_in = "hPa",
  height_in = "km"
)

sa_pressure_axis(
  name = ggplot2::waiver(),
  breaks = scales::log_breaks(n = 6),
 labels = scales::number_format(drop0trailing = TRUE, big.mark = "", trim = FALSE),
  guide = ggplot2::waiver(),
  height_in = "km",
  pressure_in = "hPa"
)
```

## Arguments

| | |
|---|---|
| height | height in meter |
| pressure | pressure in pascals |
| height_in, pressure_in | units of height and pressure, respectively. Possible values are "km", "m" for height and "hPa" and "Pa" for pressure. Alternatively, it can be a numeric constant that multiplied to convert the unit to meters and Pascals respectively. (E.g. if height is in feet, use `height_in = 0.3048`.) |
| n | desiderd number of breaks. |
| ... | extra arguments passed to scales::breaks_extended. |
| name, breaks, labels, guide | arguments passed to `ggplot2::sec_axis()` |

## Details

`sa_pressure()`, `sa_height()`, `sa_temperature()` return, respectively, pressure (in pascals), height

(in meters) and temperature (in Kelvin).

`sa_height_trans()` and `sa_pressure_trans()` are two transformation functions to be used as the `trans` argument in ggplot2 scales (e.g. scale_y_continuous(trans = "sa_height").

`sa_height_axis()` and `sa_pressure_axis()` return a secondary axis that transforms to height or pressure respectively to be used as ggplot2 secondary axis (e.g. `scale_y_continuous(sec.axis = sa_height_axis())`).

For convenience, and unlike the "primitive" functions, both the transformation functions and the axis functions input and output in hectopascals and kilometres by default.

## References

Standard atmosphere—Glossary of Meteorology. (n.d.). Retrieved 22 February 2021, from https://glossary.ametsoc.org/wiki/Standard_atmosphere

## Examples

```
height <- seq(0, 100*1000, by = 1*200)

# Temperature profile that defines the standard atmosphere (in degrees Celsius)
plot(sa_temperature(height) - 273.15, height, type = "l")

# Pressure profile
plot(sa_pressure(height), height, type = "l")

# Use with ggplot2
library(ggplot2)
data <- data.frame(height = height/1000,               # height in kilometers
                   pressure = sa_pressure(height)/100) # pressures in hectopascals

# With the sa_*_axis functions, you can label the approximate height
# when using isobaric coordinates#'
ggplot(data, aes(height, pressure)) +
  geom_path() +
  scale_y_continuous(sec.axis = sa_height_axis("height"))

# Or the approximate pressure when using physical height
ggplot(data, aes(pressure, height)) +
  geom_path() +
  scale_y_continuous(sec.axis = sa_pressure_axis("level"))

# When working with isobaric coordinates,using a linear scale exagerates
# the thickness of the lower levels
ggplot(temperature[lat == 0], aes(lon, lev)) +
   geom_contour_fill(aes(z = air)) +
   scale_y_reverse()

# Using the standard atmospehre height transormation, the result
# is an approximate linear scale in height
ggplot(temperature[lat == 0], aes(lon, lev)) +
   geom_contour_fill(aes(z = air)) +
   scale_y_continuous(trans = "sa_height", expand = c(0, 0))
```

```
# The result is very similar to using a reverse log transform, which is the
# current behaviour of scale_y_level(). This transformation slightly
# overextends the higher levels.
ggplot(temperature[lat == 0], aes(lon, lev)) +
   geom_contour_fill(aes(z = air)) +
   scale_y_level()
```

---

scale_divergent *Divergent colour scales*

---

## Description

Wrapper around ggplot's scale_colour_gradient2 with inverted defaults of high and low.

## Usage

```
scale_colour_divergent(
  ...,
  low = scales::muted("blue"),
  mid = "white",
  high = scales::muted("red"),
  midpoint = 0,
  space = "Lab",
  na.value = "grey50",
  guide = "colourbar"
)

scale_color_divergent(
  ...,
  low = scales::muted("blue"),
  mid = "white",
  high = scales::muted("red"),
  midpoint = 0,
  space = "Lab",
  na.value = "grey50",
  guide = "colourbar"
)

scale_fill_divergent(
  ...,
  low = scales::muted("blue"),
  mid = "white",
  high = scales::muted("red"),
  midpoint = 0,
  space = "Lab",
  na.value = "grey50",
```

```
  guide = "colourbar"
)
```

## Arguments

| | |
|---|---|
| `...` | Arguments passed on to `continuous_scale` |
| | `scale_name` The name of the scale that should be used for error messages associated with this scale. |
| | `palette` A palette function that when called with a numeric vector with values between 0 and 1 returns the corresponding output values (e.g., `scales::area_pal()`). |
| | `name` The name of the scale. Used as the axis or legend title. If `waiver()`, the default, the name of the scale is taken from the first mapping used for that aesthetic. If `NULL`, the legend title will be omitted. |
| | `breaks` One of:<br>• `NULL` for no breaks<br>• `waiver()` for the default breaks computed by the transformation object<br>• A numeric vector of positions<br>• A function that takes the limits as input and returns breaks as output (e.g., a function returned by `scales::extended_breaks()`). Also accepts rlang lambda function notation. |
| | `minor_breaks` One of:<br>• `NULL` for no minor breaks<br>• `waiver()` for the default breaks (one minor break between each major break)<br>• A numeric vector of positions<br>• A function that given the limits returns a vector of minor breaks. Also accepts rlang lambda function notation. |
| | `n.breaks` An integer guiding the number of major breaks. The algorithm may choose a slightly different number to ensure nice break labels. Will only have an effect if `breaks = waiver()`. Use `NULL` to use the default number of breaks given by the transformation. |
| | `labels` One of:<br>• `NULL` for no labels<br>• `waiver()` for the default labels computed by the transformation object<br>• A character vector giving labels (must be same length as `breaks`)<br>• A function that takes the breaks as input and returns labels as output. Also accepts rlang lambda function notation. |
| | `limits` One of:<br>• `NULL` to use the default scale range<br>• A numeric vector of length two providing limits of the scale. Use `NA` to refer to the existing minimum or maximum<br>• A function that accepts the existing (automatic) limits and returns new limits. Also accepts rlang lambda function notation. Note that setting limits on positional scales will **remove** data outside of the limits. If the purpose is to zoom, use the limit argument in the coordinate system (see `coord_cartesian()`). |

`rescaler` A function used to scale the input values to the range [0, 1]. This is always `scales::rescale()`, except for diverging and n colour gradients (i.e., `scale_colour_gradient2()`, `scale_colour_gradientn()`). The `rescaler` is ignored by position scales, which always use `scales::rescale()`. Also accepts rlang lambda function notation.

`oob` One of:

- Function that handles limits outside of the scale limits (out of bounds). Also accepts rlang lambda function notation.
- The default (`scales::censor()`) replaces out of bounds values with `NA`.
- `scales::squish()` for squishing out of bounds values into range.
- `scales::squish_infinite()` for squishing infinite values into range.

`trans` For continuous scales, the name of a transformation object or the object itself. Built-in transformations include "asn", "atanh", "boxcox", "date", "exp", "hms", "identity", "log", "log10", "log1p", "log2", "logit", "modulus", "probability", "probit", "pseudo_log", "reciprocal", "reverse", "sqrt" and "time".

A transformation object bundles together a transform, its inverse, and methods for generating breaks and labels. Transformation objects are defined in the scales package, and are called <name>_trans (e.g., `scales::boxcox_trans()`). You can create your own transformation with `scales::trans_new()`.

`expand` For position scales, a vector of range expansion constants used to add some padding around the data to ensure that they are placed some distance away from the axes. Use the convenience function `expansion()` to generate the values for the `expand` argument. The defaults are to expand the scale by 5% on each side for continuous variables, and by 0.6 units on each side for discrete variables.

`position` For position scales, The position of the axis. `left` or `right` for `y` axes, `top` or `bottom` for x axes.

`super` The super class to use for the constructed scale

`low` Colours for low and high ends of the gradient.

`mid` colour for mid point

`high` Colours for low and high ends of the gradient.

`midpoint` The midpoint (in data value) of the diverging scale. Defaults to 0.

`space` colour space in which to calculate gradient. Must be "Lab" - other values are deprecated.

`na.value` Colour to use for missing values

`guide` Type of legend. Use `"colourbar"` for continuous colour bar, or `"legend"` for discrete colour legend.

## See Also

Other ggplot2 helpers: `DivideTimeseries()`, `MakeBreaks()`, `WrapCircular()`, `geom_arrow()`, `geom_contour2()`, `geom_contour_fill()`, `geom_label_contour()`, `geom_relief()`, `geom_streamline()`, `guide_colourstrip()`, `map_labels`, `reverselog_trans()`, `scale_longitude`, `stat_na()`, `stat_subset()`

## Examples

```
library(ggplot2)
ggplot(reshape2::melt(volcano), aes(Var1, Var2, z = value)) +
  geom_contour(aes(color = ..level..)) +
  scale_colour_divergent(midpoint = 130)
```

---

`scale_fill_discretised` *Discretised scale*

---

## Description

This scale allows ggplot to understand data that has been discretised with some procedure akin to `cut` and access the underlying continuous values. For a scale that does the opposite (take continuous data and treat them as discrete) see `ggplot2::binned_scale()`.

## Usage

```
scale_fill_discretised(
  ...,
  low = "#132B43",
  high = "#56B1F7",
  space = "Lab",
  na.value = "grey50",
  guide = ggplot2::guide_colorsteps(even.steps = FALSE, show.limits = TRUE),
  aesthetics = "fill"
)

scale_fill_divergent_discretised(
  ...,
  low = scales::muted("blue"),
  mid = "white",
  high = scales::muted("red"),
  midpoint = 0,
  space = "Lab",
  na.value = "grey50",
  guide = ggplot2::guide_colorsteps(even.steps = FALSE, show.limits = TRUE)
)

discretised_scale(
  aesthetics,
  scale_name,
  palette,
  name = ggplot2::waiver(),
  breaks = ggplot2::waiver(),
  labels = ggplot2::waiver(),
```

```
  limits = NULL,
  trans = scales::identity_trans(),
  na.value = NA,
  drop = FALSE,
  guide = ggplot2::guide_colorsteps(even.steps = FALSE),
  position = "left",
  rescaler = scales::rescale,
  oob = scales::censor,
  super = ScaleDiscretised
)
```

## Arguments

`...` Arguments passed on to `continuous_scale`

`scale_name` The name of the scale that should be used for error messages associated with this scale.

`palette` A palette function that when called with a numeric vector with values between 0 and 1 returns the corresponding output values (e.g., `scales::area_pal()`).

`name` The name of the scale. Used as the axis or legend title. If `waiver()`, the default, the name of the scale is taken from the first mapping used for that aesthetic. If `NULL`, the legend title will be omitted.

`breaks` One of:

- `NULL` for no breaks
- `waiver()` for the default breaks computed by the transformation object
- A numeric vector of positions
- A function that takes the limits as input and returns breaks as output (e.g., a function returned by `scales::extended_breaks()`). Also accepts rlang lambda function notation.

`minor_breaks` One of:

- `NULL` for no minor breaks
- `waiver()` for the default breaks (one minor break between each major break)
- A numeric vector of positions
- A function that given the limits returns a vector of minor breaks. Also accepts rlang lambda function notation.

`n.breaks` An integer guiding the number of major breaks. The algorithm may choose a slightly different number to ensure nice break labels. Will only have an effect if `breaks = waiver()`. Use `NULL` to use the default number of breaks given by the transformation.

`labels` One of:

- `NULL` for no labels
- `waiver()` for the default labels computed by the transformation object
- A character vector giving labels (must be same length as `breaks`)
- A function that takes the breaks as input and returns labels as output. Also accepts rlang lambda function notation.

`limits` One of:

- NULL to use the default scale range
- A numeric vector of length two providing limits of the scale. Use NA to refer to the existing minimum or maximum
- A function that accepts the existing (automatic) limits and returns new limits. Also accepts rlang lambda function notation. Note that setting limits on positional scales will **remove** data outside of the limits. If the purpose is to zoom, use the limit argument in the coordinate system (see coord_cartesian()).

rescaler A function used to scale the input values to the range [0, 1]. This is always scales::rescale(), except for diverging and n colour gradients (i.e., scale_colour_gradient2(), scale_colour_gradientn()). The rescaler is ignored by position scales, which always use scales::rescale(). Also accepts rlang lambda function notation.

oob One of:

- Function that handles limits outside of the scale limits (out of bounds). Also accepts rlang lambda function notation.
- The default (scales::censor()) replaces out of bounds values with NA.
- scales::squish() for squishing out of bounds values into range.
- scales::squish_infinite() for squishing infinite values into range.

trans For continuous scales, the name of a transformation object or the object itself. Built-in transformations include "asn", "atanh", "boxcox", "date", "exp", "hms", "identity", "log", "log10", "log1p", "log2", "logit", "modulus", "probability", "probit", "pseudo_log", "reciprocal", "reverse", "sqrt" and "time".

A transformation object bundles together a transform, its inverse, and methods for generating breaks and labels. Transformation objects are defined in the scales package, and are called <name>_trans (e.g., scales::boxcox_trans()). You can create your own transformation with scales::trans_new().

expand For position scales, a vector of range expansion constants used to add some padding around the data to ensure that they are placed some distance away from the axes. Use the convenience function expansion() to generate the values for the expand argument. The defaults are to expand the scale by 5% on each side for continuous variables, and by 0.6 units on each side for discrete variables.

position For position scales, The position of the axis. left or right for y axes, top or bottom for x axes.

super The super class to use for the constructed scale

low Colours for low and high ends of the gradient.

high Colours for low and high ends of the gradient.

space colour space in which to calculate gradient. Must be "Lab" - other values are deprecated.

na.value Colour to use for missing values

guide Type of legend. Use "colourbar" for continuous colour bar, or "legend" for discrete colour legend.

`aesthetics` Character string or vector of character strings listing the name(s) of the aesthetic(s) that this scale works with. This can be useful, for example, to apply colour settings to the `colour` and `fill` aesthetics at the same time, via `aesthetics = c("colour","fill")`.

`mid` colour for mid point

`midpoint` The midpoint (in data value) of the diverging scale. Defaults to 0.

`scale_name` The name of the scale that should be used for error messages associated with this scale.

`palette` A palette function that when called with a numeric vector with values between 0 and 1 returns the corresponding output values (e.g., `scales::area_pal()`).

`name` The name of the scale. Used as the axis or legend title. If `waiver()`, the default, the name of the scale is taken from the first mapping used for that aesthetic. If `NULL`, the legend title will be omitted.

`breaks` One of:

- `NULL` for no breaks
- `waiver()` for the default breaks computed by the transformation object
- A numeric vector of positions
- A function that takes the limits as input and returns breaks as output (e.g., a function returned by `scales::extended_breaks()`). Also accepts rlang lambda function notation.

`labels` One of:

- `NULL` for no labels
- `waiver()` for the default labels computed by the transformation object
- A character vector giving labels (must be same length as `breaks`)
- A function that takes the breaks as input and returns labels as output. Also accepts rlang lambda function notation.

`limits` One of:

- `NULL` to use the default scale range
- A numeric vector of length two providing limits of the scale. Use `NA` to refer to the existing minimum or maximum
- A function that accepts the existing (automatic) limits and returns new limits. Also accepts rlang lambda function notation. Note that setting limits on positional scales will **remove** data outside of the limits. If the purpose is to zoom, use the limit argument in the coordinate system (see `coord_cartesian()`).

`trans` For continuous scales, the name of a transformation object or the object itself. Built-in transformations include "asn", "atanh", "boxcox", "date", "exp", "hms", "identity", "log", "log10", "log1p", "log2", "logit", "modulus", "probability", "probit", "pseudo_log", "reciprocal", "reverse", "sqrt" and "time".

A transformation object bundles together a transform, its inverse, and methods for generating breaks and labels. Transformation objects are defined in the scales package, and are called <name>_trans (e.g., `scales::boxcox_trans()`). You can create your own transformation with `scales::trans_new()`.

drop — Should unused factor levels be omitted from the scale? The default, TRUE, uses the levels that appear in the data; FALSE uses all the levels in the factor.

position — For position scales, The position of the axis. `left` or `right` for y axes, `top` or `bottom` for x axes.

rescaler — A function used to scale the input values to the range [0, 1]. This is always `scales::rescale()`, except for diverging and n colour gradients (i.e., `scale_colour_gradient2()`, `scale_colour_gradientn()`). The `rescaler` is ignored by position scales, which always use `scales::rescale()`. Also accepts rlang lambda function notation.

oob — One of:

- Function that handles limits outside of the scale limits (out of bounds). Also accepts rlang lambda function notation.
- The default (`scales::censor()`) replaces out of bounds values with `NA`.
- `scales::squish()` for squishing out of bounds values into range.
- `scales::squish_infinite()` for squishing infinite values into range.

super — The super class to use for the constructed scale

## Details

This scale makes it very easy to synchronise the breaks of filled contours and the breaks shown no the colour guide. Bear in mind that when using `geom_contour_fill()`, the default fill aesthetic (`level_mid`) is **not** discretised. To use this scale with that geom, you need to set `aes(fill = stat(level))`.

## Examples

```
library(ggplot2)

# Using the `level` compute aesthetic from `geom_contour_fill()`
# (or ggplot2::geom_contour_filled()), the default scale is discrete.
# This means that you cannot map colours to the underying numbers.
v <- ggplot(faithfuld, aes(waiting, eruptions, z = density))
v + geom_contour_fill(aes(fill = stat(level)))

v + geom_contour_fill(aes(fill = stat(level))) +
  scale_fill_discretised()

# The scale can be customised the same as any continuous colour scale
v + geom_contour_fill(aes(fill = stat(level))) +
  scale_fill_discretised(low = "#a62100", high = "#fff394")

# Setting limits explicitly will truncate the scale
# (if any limit is inside the range of the breaks but doesn't
# coincide with any range, it will be rounded with a warning)
v + geom_contour_fill(aes(fill = stat(level))) +
  scale_fill_discretised(low = "#a62100", high = "#fff394",
                         limits = c(0.01, 0.028))

# Or extend it.
```

```
v + geom_contour_fill(aes(fill = stat(level))) +
  scale_fill_discretised(low = "#a62100", high = "#fff394",
                         limits = c(0, 0.07))

v + geom_contour_fill(aes(fill = stat(level))) +
  scale_fill_divergent_discretised(midpoint = 0.02)

# Existing continous scales can be "retrofitted" by changing the `super`
# and `guide` arguments.
v + geom_contour_fill(aes(fill = stat(level))) +
    scale_fill_distiller(super = ScaleDiscretised)

# Unequal breaks will, by default, map to unequal spacing in the guide
v + geom_contour_fill(aes(fill = stat(level)), breaks = c(0, 0.005, 0.01, 0.02, 0.04)) +
  scale_fill_discretised()

# You can change that by the `even.steps` argument on ggplot2::guide_colorsteps()
v + geom_contour_fill(aes(fill = stat(level)), breaks = c(0, 0.005, 0.01, 0.02, 0.04)) +
  scale_fill_discretised(guide = guide_colorsteps(even.steps = TRUE, show.limits = TRUE))
```

---

`scale_label_colour_continuous` *Scales for contour label aesthetics*

---

## Description

Scales for contour label aesthetics

## Usage

```
scale_label_colour_continuous(
  ...,
  aesthetics = c("label_colour"),
  guide = ggplot2::guide_colorbar(available_aes = "label_colour")
)

scale_label_alpha_continuous(
  ...,
  range = c(0.1, 1),
  aesthetics = c("label_alpha")
)

scale_label_size_continuous(
  name = waiver(),
  breaks = waiver(),
  labels = waiver(),
  limits = NULL,
```

```
  range = c(1, 6),
  trans = "identity",
  guide = "legend"
)
```

## Arguments

| | |
|---|---|
| ... | Arguments passed on to `continuous_scale`<br>`minor_breaks` One of:<br>• `NULL` for no minor breaks<br>• `waiver()` for the default breaks (one minor break between each major break)<br>• A numeric vector of positions<br>• A function that given the limits returns a vector of minor breaks. Also accepts rlang lambda function notation.<br>`oob` One of:<br>• Function that handles limits outside of the scale limits (out of bounds). Also accepts rlang lambda function notation.<br>• The default (`scales::censor()`) replaces out of bounds values with `NA`.<br>• `scales::squish()` for squishing out of bounds values into range.<br>• `scales::squish_infinite()` for squishing infinite values into range.<br>`na.value` Missing values will be replaced with this value.<br>`expand` For position scales, a vector of range expansion constants used to add some padding around the data to ensure that they are placed some distance away from the axes. Use the convenience function `expansion()` to generate the values for the `expand` argument. The defaults are to expand the scale by 5% on each side for continuous variables, and by 0.6 units on each side for discrete variables.<br>`position` For position scales, The position of the axis. `left` or `right` for `y` axes, `top` or `bottom` for x axes.<br>`super` The super class to use for the constructed scale |
| `aesthetics` | Character string or vector of character strings listing the name(s) of the aesthetic(s) that this scale works with. This can be useful, for example, to apply colour settings to the colour and fill aesthetics at the same time, via aesthetics = c("colour", "fill"). |
| `guide` | Type of legend. Use "colourbar" for continuous colour bar, or "legend" for discrete colour legend. |
| `range` | Output range of alpha values. Must lie between 0 and 1. |
| `name` | The name of the scale. Used as the axis or legend title. If `waiver()`, the default, the name of the scale is taken from the first mapping used for that aesthetic. If `NULL`, the legend title will be omitted. |
| `breaks` | One of:<br>• `NULL` for no breaks |

- `waiver()` for the default breaks computed by the transformation object
- A numeric vector of positions
- A function that takes the limits as input and returns breaks as output (e.g., a function returned by `scales::extended_breaks()`). Also accepts rlang lambda function notation.

`labels` One of:

- `NULL` for no labels
- `waiver()` for the default labels computed by the transformation object
- A character vector giving labels (must be same length as `breaks`)
- A function that takes the breaks as input and returns labels as output. Also accepts rlang lambda function notation.

`limits` One of:

- `NULL` to use the default scale range
- A numeric vector of length two providing limits of the scale. Use `NA` to refer to the existing minimum or maximum
- A function that accepts the existing (automatic) limits and returns new limits. Also accepts rlang lambda function notation. Note that setting limits on positional scales will **remove** data outside of the limits. If the purpose is to zoom, use the limit argument in the coordinate system (see `coord_cartesian()`).

`trans` For continuous scales, the name of a transformation object or the object itself. Built-in transformations include "asn", "atanh", "boxcox", "date", "exp", "hms", "identity", "log", "log10", "log1p", "log2", "logit", "modulus", "probability", "probit", "pseudo_log", "reciprocal", "reverse", "sqrt" and "time".

A transformation object bundles together a transform, its inverse, and methods for generating breaks and labels. Transformation objects are defined in the scales package, and are called <name>_trans (e.g., `scales::boxcox_trans()`). You can create your own transformation with `scales::trans_new()`.

---

`scale_longitude` *Helpful scales for maps*

---

## Description

These functions are simple wrappers around `scale_x_continuous` and `scale_y_continuous` with helpful defaults for plotting longitude, latitude and pressure levels.

## Usage

```
scale_x_longitude(
  name = "",
  ticks = 30,
  breaks = seq(-180, 360, by = ticks),
  expand = c(0, 0),
```

```
  labels = LonLabel,
  trans = "identity",
  ...
)

scale_y_longitude(
  name = "",
  ticks = 60,
  breaks = seq(-180, 360, by = ticks),
  expand = c(0, 0),
  labels = LonLabel,
  trans = "identity",
  ...
)

scale_x_latitude(
  name = "",
  ticks = 30,
  breaks = seq(-90, 90, by = ticks),
  expand = c(0, 0),
  labels = LatLabel,
  ...
)

scale_y_latitude(
  name = "",
  ticks = 30,
  breaks = seq(-90, 90, by = ticks),
  expand = c(0, 0),
  labels = LatLabel,
  ...
)

scale_x_level(name = "", expand = c(0, 0), trans = "sa_height", ...)

scale_y_level(name = "", expand = c(0, 0), trans = "sa_height", ...)
```

## Arguments

| | |
|---|---|
| name | The name of the scale. Used as the axis or legend title. If `waiver()`, the default, the name of the scale is taken from the first mapping used for that aesthetic. If `NULL`, the legend title will be omitted. |
| ticks | spacing between breaks |
| breaks | One of:<br>• `NULL` for no breaks<br>• `waiver()` for the default breaks computed by the transformation object<br>• A numeric vector of positions |

- A function that takes the limits as input and returns breaks as output (e.g., a function returned by `scales::extended_breaks()`). Also accepts rlang lambda function notation.

`expand` For position scales, a vector of range expansion constants used to add some padding around the data to ensure that they are placed some distance away from the axes. Use the convenience function `expansion()` to generate the values for the `expand` argument. The defaults are to expand the scale by 5% on each side for continuous variables, and by 0.6 units on each side for discrete variables.

`labels` One of:

- `NULL` for no labels
- `waiver()` for the default labels computed by the transformation object
- A character vector giving labels (must be same length as `breaks`)
- A function that takes the breaks as input and returns labels as output. Also accepts rlang lambda function notation.

`trans` For continuous scales, the name of a transformation object or the object itself. Built-in transformations include "asn", "atanh", "boxcox", "date", "exp", "hms", "identity", "log", "log10", "log1p", "log2", "logit", "modulus", "probability", "probit", "pseudo_log", "reciprocal", "reverse", "sqrt" and "time".

A transformation object bundles together a transform, its inverse, and methods for generating breaks and labels. Transformation objects are defined in the scales package, and are called <name>_trans (e.g., `scales::boxcox_trans()`). You can create your own transformation with `scales::trans_new()`.

`...` Other arguments passed on to scale_(xly)_continuous()

## See Also

Other ggplot2 helpers: `DivideTimeseries()`, `MakeBreaks()`, `WrapCircular()`, `geom_arrow()`, `geom_contour2()`, `geom_contour_fill()`, `geom_label_contour()`, `geom_relief()`, `geom_streamline()`, `guide_colourstrip()`, `map_labels`, `reverselog_trans()`, `scale_divergent`, `stat_na()`, `stat_subset()`

## Examples

```
data(geopotential)
library(ggplot2)
ggplot(geopotential[date == date[1]], aes(lon, lat, z = gh)) +
    geom_contour() +
    scale_x_longitude() +
    scale_y_latitude()

data(temperature)
ggplot(temperature[lon == lon[1] & lat == lat[1]], aes(air, lev)) +
    geom_path() +
    scale_y_level()
```

scale_mag *Scale for vector magnitudes*

## Description

Allows to control the size of the arrows in geom_arrow. Highly experimental.

## Usage

```
scale_mag(
  name = ggplot2::waiver(),
  labels = ggplot2::waiver(),
  max_size = 1,
  default_unit = "cm",
  max = ggplot2::waiver(),
  guide = guide_vector(),
  ...
)

scale_mag_continuous(
  name = ggplot2::waiver(),
  labels = ggplot2::waiver(),
  max_size = 1,
  default_unit = "cm",
  max = ggplot2::waiver(),
  guide = guide_vector(),
  ...
)
```

## Arguments

| | |
|---|---|
| name | The name of the scale. Used as the axis or legend title. If `waiver()`, the default, the name of the scale is taken from the first mapping used for that aesthetic. If `NULL`, the legend title will be omitted. |
| labels | One of:<br>• `NULL` for no labels<br>• `waiver()` for the default labels computed by the transformation object<br>• A character vector giving labels (must be same length as `breaks`)<br>• A function that takes the breaks as input and returns labels as output. Also accepts rlang lambda function notation. |
| max_size | size of the arrow in centimetres |
| default_unit | ignored |
| max | magnitude of the reference arrow in data units. Will be the maximum value if `waiver()` |
| guide | type of legend |
| ... | Other arguments passed on to scale_(x\|y)_continuous() |

## Examples

```
library(ggplot2)
g <- ggplot(seals, aes(long, lat)) +
    geom_vector(aes(dx = delta_long, dy = delta_lat), skip = 2)

g + scale_mag("Seals velocity")

g + scale_mag("Seals velocity", max = 1)

g + scale_mag("Seals velocity", max_size = 2)
g + scale_mag("Seals velocity", default_unit = "mm")
```

---

# season *Assign seasons to months*

---

## Description

Assign seasons to months

## Usage

```
season(x, lang = c("en", "es"))

seasonally(x)

is.full_season(x)
```

## Arguments

| | |
|---|---|
| x | A vector of dates (alternative a numeric vector of months, for `season()`) |
| lang | Language to use. |

## Value

`season()` returns a factor vector of the same length as `x` with the trimester of each month. `seasonaly()` returns a date vector of the same length as `x` with the date "rounded" up to the centre month of each season. `is.full_season()` returns a logical vector of the same length as `x` that is true only if the 3 months of each season for each year (December counts for the following year) are present in the dataset.

## Examples

```
season(1, lang = "en")
season(as.Date("2017-01-01"))

seasonally(as.Date(c("2017-12-01", "2018-01-01", "2018-02-01")))

is.full_season(as.Date(c("2017-12-01", "2018-01-01", "2018-02-01", "2018-03-01")))
```

## spherical *Transform between spherical coordinates and physical coordinates*

### Description

Transform a longitude or latitude interval into the equivalent in meters depending on latitude.

### Usage

```
dlon(dx, lat, a = 6731000)

dlat(dy, a = 6731000)

dx(dlon, lat, a = 6731000)

dy(dlat, a = 6731000)
```

### Arguments

| | |
|---|---|
| dx, dy | interval in meters |
| lat | latitude, in degrees |
| a | radius of the Earth |
| dlon, dlat | interval in degrees |

### Examples

```
library(data.table)
data(geopotential)
geopotential <- geopotential[date == date[1]]

# Geostrophic wind
geopotential[, c("u", "v") := GeostrophicWind(gh, lon, lat)]  # in meters/second
geopotential[, c("dlon", "dlat") := .(dlon(u, lat), dlat(v))] # in degrees/second
geopotential[, c("u2", "v2") := .(dx(dlon, lat), dy(dlat))]   # again in degrees/second
```

## stat_na *Filter only NA values.*

### Description

Useful for indicating or masking missing data. This stat subsets data where one variable is NA.

## Usage

```
stat_na(
  mapping = NULL,
  data = NULL,
  geom = "point",
  position = "identity",
  ...,
  show.legend = NA,
  inherit.aes = TRUE
)
```

## Arguments

| | |
|---|---|
| mapping | Set of aesthetic mappings created by aes() or aes_(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping. |
| data | The data to be displayed in this layer. There are three options:<br>If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot().<br>A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created.<br>A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x,10)). |
| geom | The geometric object to use display the data |
| position | Position adjustment, either as a string, or the result of a call to a position adjustment function. |
| ... | Other arguments passed on to layer(). These are often aesthetics, used to set an aesthetic to a fixed value, like colour = "red" or size = 3. They may also be parameters to the paired geom/stat. |
| show.legend | logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display. |
| inherit.aes | If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders(). |

## Aesthetics

stat_na understands the following aesthetics (required aesthetics are in bold)

- **x**
- **y**
- **na**
- width
- height

## See Also

stat_subset for a more general way of filtering data.

Other ggplot2 helpers: DivideTimeseries(), MakeBreaks(), WrapCircular(), geom_arrow(), geom_contour2(), geom_contour_fill(), geom_label_contour(), geom_relief(), geom_streamline(), guide_colourstrip(), map_labels, reverselog_trans(), scale_divergent, scale_longitude, stat_subset()

## Examples

```
library(ggplot2)
library(data.table)
surface <- reshape2::melt(volcano)
surface <- within(surface, value[Var1 %between% c(20, 30) & Var2 %between% c(20, 30)] <- NA)
surface[sample(1:nrow(surface), 100, replace = FALSE), 3] <- NA

ggplot(surface, aes(Var1, Var2, z = value)) +
    geom_contour_fill(na.fill = TRUE) +
    stat_na(aes(na = value), geom = "tile")
```

---

# stat_subset *Subset values*

---

## Description

Removes values where subset evaluates to FALSE. Useful for showing only statistical significant values, or an interesting subset of the data without manually subsetting the data.

## Usage

```
stat_subset(
  mapping = NULL,
  data = NULL,
  geom = "point",
  position = "identity",
  ...,
  show.legend = NA,
  inherit.aes = TRUE
)
```

## Arguments

| | |
|---|---|
| mapping | Set of aesthetic mappings created by aes() or aes_(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping. |

| | |
|---|---|
| data | The data to be displayed in this layer. There are three options:<br>If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot().<br>A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created.<br>A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x,10)). |
| geom | The geometric object to use display the data |
| position | Position adjustment, either as a string, or the result of a call to a position adjustment function. |
| ... | Other arguments passed on to layer(). These are often aesthetics, used to set an aesthetic to a fixed value, like colour = "red" or size = 3. They may also be parameters to the paired geom/stat. |
| show.legend | logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display. |
| inherit.aes | If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders(). |

## Aesthetics

stat_subset understands the following aesthetics (required aesthetics are in bold)

- **x**
- **y**
- **subset**
- width
- height

## See Also

stat_na for a more specialized stat for filtering NA values.

Other ggplot2 helpers: DivideTimeseries(), MakeBreaks(), WrapCircular(), geom_arrow(), geom_contour2(), geom_contour_fill(), geom_label_contour(), geom_relief(), geom_streamline(), guide_colourstrip(), map_labels, reverselog_trans(), scale_divergent, scale_longitude, stat_na()

## Examples

```
library(ggplot2)
ggplot(reshape2::melt(volcano), aes(Var1, Var2)) +
    geom_contour(aes(z = value)) +
```

```
  stat_subset(aes(subset = value >= 150 & value <= 160),
              shape = 3, color = "red")
```

---

`surface` *Surface height*

---

## Description

Surface height of central Argentina on a lambert grid.

## Usage

```
surface
```

## Format

A data.table with 53224 rows and 5 variables.

**lon** longitude in degrees

**lat** latitude in degrees

**height** height in meters

**x** x coordinates of projection

**y** y coordinates of projection

---

`temperature` *Air temperature*

---

## Description

A global air temperature field for 2017-07-09.

## Usage

```
temperature
```

## Format

A data.table with 10512 rows and 3 variables:

**lon** longitude in degrees from 0 to 360

**lat** latitude in degrees

**lev** pressure level in hPa)

**air** air temperature in Kelvin

## Source

https://psl.noaa.gov/data/gridded/data.ncep.reanalysis.derived.pressure.html

`thermodynamics` *Thermodynamics*

## Description

Functions related to common atmospheric thermodynamic relationships.

## Usage

```
IdealGas(p, t, rho, R = 287.058)

Adiabat(p, t, theta, p0 = 1e+05, kappa = 2/7)

VirtualTemperature(p, t, e, tv, epsilon = 0.622)

MixingRatio(p, e, w, epsilon = 0.622)

ClausiusClapeyron(t, es)

DewPoint(p, ws, td, epsilon = 0.622)
```

## Arguments

| | |
|---|---|
| `p` | pressure |
| `t` | temperature |
| `rho` | density |
| `R` | gas constant for air |
| `theta` | potential temperature |
| `p0` | reference pressure |
| `kappa` | ratio of dry air constant and specific heat capacity at constant pressure |
| `e` | vapour partial pressure |
| `tv` | virtual temperature |
| `epsilon` | ratio of dry air constant and vapour constant |
| `w` | mixing ratio |
| `es` | saturation vapour partial pressure |
| `ws` | saturation mixing ratio |
| `td` | dewpoint |

## Details

`IdealGas` computes pressure, temperature or density of air according to the ideal gas law $P = \rho RT$.

`Adiabat` computes pressure, temperature or potential temperature according to the adiabatic relationship $\theta = T(P0/P)^{\kappa}$.

`VirtualTemperature` computes pressure, temperature, vapour partial pressure or virtual temperature according to the virtual temperature definition $T(1 - e/P(1 - \epsilon))^{-1}$.

`MixingRatio` computes pressure, vapour partial temperature, or mixing ratio according to $w = \epsilon e/(P - e)$.

`ClausiusClapeyron` computes saturation pressure or temperature according to the August-Roche-Magnus formula $es = aexpbT/(T + c)$ with temperature in Kelvin and saturation pressure in Pa.

`DewPoint` computes pressure, saturation mixing ration or dew point from the relationship $ws = \epsilon es(Td)/(p - es(Td))$. Note that the computation of dew point is approximated.

Is important to take note of the units in which each variable is provided. With the default values, pressure should be passed in Pascals, temperature and potential temperature in Kelvins, and density in $kg/m^3$. `ClausiusClayperon` and `DewPoint` require and return values in those units.

The defaults value of the `R` and `kappa` parameters are correct for dry air, for the case of moist air, use the virtual temperature instead of the actual temperature.

## Value

Each function returns the value of the missing state variable.

## References

http://www.atmo.arizona.edu/students/courselinks/fall11/atmo551a/ATMO_451a_551a_files/WaterVapor.pdf

## See Also

Other meteorology functions: `Derivate()`, `EOF()`, `GeostrophicWind()`, `WaveFlux()`, `waves`

## Examples

```
IdealGas(1013*100, 20 + 273.15)
IdealGas(1013*100, rho = 1.15) - 273.15

(theta <- Adiabat(70000, 20 + 273.15))
Adiabat(70000, theta = theta) - 273.15

# Relative humidity from T and Td
t <- 25 + 273.15
td <- 20 + 273.15
p <- 1000000
(rh <- ClausiusClapeyron(td)/ClausiusClapeyron(t))

# Mixing ratio
ws <- MixingRatio(p, ClausiusClapeyron(t))
w <- ws*rh
DewPoint(p, w) - 273.15    # Recover Td
```

## Trajectory *Compute trajectories*

### Description

Computes trajectories of particles in a time-varying velocity field.

### Usage

```
Trajectory(formula, x0, y0, cyclical = FALSE, data = NULL, res = 2)
```

### Arguments

| | |
|---|---|
| formula | a formula indicating dependent and independent variables in the form of dx + dy ~ x + y + t. |
| x0, y0 | starting coordinates of the particles. |
| cyclical | logical vector of boundary condition for x and y. |
| data | optional data.frame containing the variables. |
| res | resolution parameter (higher numbers increases the resolution) |

## WaveFlux *Calculate wave-activity flux*

### Description

Calculate wave-activity flux

### Usage

```
WaveFlux(gh, u, v, lon, lat, lev, g = 9.81, a = 6371000)
```

### Arguments

| | |
|---|---|
| gh | geopotential height |
| u | mean zonal velocity |
| v | mean meridional velocity |
| lon | longitude (in degrees) |
| lat | latitude (in degrees) |
| lev | pressure level (in hPa) |
| g | acceleration of gravity |
| a | Earth's radius |

## Details

Calculates Plum-like wave activity fluxes

## Value

A list with elements: longitude, latitude, and the two horizontal components of the wave activity flux.

## References

Takaya, K. and H. Nakamura, 2001: A Formulation of a Phase-Independent Wave-Activity Flux for Stationary and Migratory Quasigeostrophic Eddies on a Zonally Varying Basic Flow. J. Atmos. Sci., 58, 608–627, doi: 10.1175/15200469(2001)058<0608:AFOAPI>2.0.CO;2
Adapted from https://github.com/marisolosman/Reunion_Clima/blob/master/WAF/Calculo_WAF.ipynb

## See Also

Other meteorology functions: Derivate(), EOF(), GeostrophicWind(), thermodynamics, waves

---

# waves *Fourier transform*

---

## Description

Perform a fourier transform of the data and return the

## Usage

```
FitWave(y, k = 1)

BuildWave(
  x,
  amplitude,
  phase,
  k,
  wave = list(amplitude = amplitude, phase = phase, k = k),
  sum = TRUE
)

FilterWave(y, k, action = sign(k[k != 0][1]))

WaveEnvelope(y)
```

## Arguments

| | |
|---|---|
| y | numeric vector to transform |
| k | numeric vector of wave numbers |
| x | numeric vector of locations (in radians) |
| amplitude | numeric vector of amplitudes |
| phase | numeric vector of phases |
| wave | optional list output from FitWave |
| sum | whether to perform the sum or not (see Details) |
| action | integer to disambiguate action for k = 0 (see Details) |

## Details

FitWave uses fft to make a fourier transform of the data and then returns a list of parameters for each wave number kept. The amplitude (A), phase ($\phi$) and wave number (k) satisfy:

$$y = \sum Acos((x - \phi)k)$$

The phase is calculated so that it lies between 0 and $2\pi/k$ so it represents the location (in radians) of the first maximum of each wave number. For the case of k = 0 (the mean), phase is arbitrarily set to 0.

BuildWave is FitWave's inverse. It reconstructs the original data for selected wavenumbers. If sum is TRUE (the default) it performs the above mentioned sum and returns a single vector. If is FALSE, then it returns a list of k vectors consisting of the reconstructed signal of each wavenumber.

FilterWave filters or removes wavenumbers specified in k. If k is positive, then the result is the reconstructed signal of y only for wavenumbers specified in k, if it's negative, is the signal of y minus the wavenumbers specified in k. The argument action must be be manually set to -1 or +1 if k=0.

WaveEnvelope computes the wave envelope of y following Zimin (2003). To compute the envelope of only a restricted band, first filter it with FilterWave.

## Value

FitWaves returns a a named list with components

**k** wavenumbers

**amplitude** amplitude of each wavenumber

**phase** phase of each wavenumber in radians

**r2** explained variance of each wavenumber

BuildWave returns a vector of the same length of x with the reconstructed vector if sum is TRUE or, instead, a list with components

**k** wavenumbers

**x** the vector of locations

**y** the reconstructed signal of each wavenumber

FilterWave returns a vector of the same length as y '

## References

Zimin, A.V., I. Szunyogh, D.J. Patil, B.R. Hunt, and E. Ott, 2003: Extracting Envelopes of Rossby Wave Packets. Mon. Wea. Rev., 131, 1011–1017, doi: 10.1175/15200493(2003)131<1011:EEORWP>2.0.CO;2

## See Also

Other meteorology functions: `Derivate()`, `EOF()`, `GeostrophicWind()`, `WaveFlux()`, `thermodynamics`

## Examples

```
data(geopotential)
library(data.table)
# January mean of geopotential height
jan <- geopotential[month(date) == 1, .(gh = mean(gh)), by = .(lon, lat)]

# Stationary waves for each latitude
jan.waves <- jan[, FitWave(gh, 1:4), by = .(lat)]
library(ggplot2)
ggplot(jan.waves, aes(lat, amplitude, color = factor(k))) +
    geom_line()

# Build field of wavenumber 1
jan[, gh.1 := BuildWave(lon*pi/180, wave = FitWave(gh, 1)), by = .(lat)]
ggplot(jan, aes(lon, lat)) +
    geom_contour(aes(z = gh.1, color = ..level..)) +
    coord_polar()

# Build fields of wavenumber 1 and 2
waves <- jan[, BuildWave(lon*pi/180, wave = FitWave(gh, 1:2), sum = FALSE), by = .(lat)]
waves[, lon := x*180/pi]
ggplot(waves, aes(lon, lat)) +
    geom_contour(aes(z = y, color = ..level..)) +
    facet_wrap(~k) +
    coord_polar()

# Field with waves 0 to 2 filtered
jan[, gh.no12 := gh - BuildWave(lon*pi/180, wave = FitWave(gh, 0:2)), by = .(lat)]
ggplot(jan, aes(lon, lat)) +
    geom_contour(aes(z = gh.no12, color = ..level..)) +
    coord_polar()

# Much faster
jan[, gh.no12 := FilterWave(gh, -2:0), by = .(lat)]
ggplot(jan, aes(lon, lat)) +
    geom_contour(aes(z = gh.no12, color = ..level..)) +
    coord_polar()

# Using positive numbers returns the field
jan[, gh.only12 := FilterWave(gh, 2:1), by = .(lat)]
ggplot(jan, aes(lon, lat)) +
    geom_contour(aes(z = gh.only12, color = ..level..)) +
    coord_polar()
```

```
# Compute the envelope of the geopotential
jan[, envelope := WaveEnvelope(gh.no12), by = .(lat)]
ggplot(jan[lat == -60], aes(lon, gh.no12)) +
    geom_line() +
    geom_line(aes(y = envelope), color = "red")
```

---

`WrapCircular` *Wrap periodic data to any range*

---

## Description

Periodic data can be defined only in one period and be extended to any arbitrary range.

## Usage

```
WrapCircular(x, circular = "lon", wrap = c(0, 360))
```

## Arguments

| | |
|---|---|
| `x` | a data.frame |
| `circular` | the name of the circular dimension |
| `wrap` | the wrap for the data to be extended to |

## Value

A data.frame.

## See Also

geom_contour2

Other ggplot2 helpers: `DivideTimeseries()`, `MakeBreaks()`, `geom_arrow()`, `geom_contour2()`, `geom_contour_fill()`, `geom_label_contour()`, `geom_relief()`, `geom_streamline()`, `guide_colourstrip()`, `map_labels`, `reverselog_trans()`, `scale_divergent`, `scale_longitude`, `stat_na()`, `stat_subset()`

## Examples

```
library(ggplot2)
library(data.table)
data(geopotential)
g <- ggplot(geopotential[date == date[1]], aes(lon, lat)) +
    geom_contour(aes(z = gh)) +
    coord_polar() +
    ylim(c(-90, -10))

# This plot has problems in lon = 0
g
```

```
# But using WrapCircular solves it.
g %+% WrapCircular(geopotential[date == date[1]], "lon", c(0, 360))

# Aditionally data can be just repeatet to the right and
# left
ggplot(WrapCircular(geopotential[date == date[1]], wrap = c(-180, 360 + 180)),
       aes(lon, lat)) +
    geom_contour(aes(z = gh))

# The same behaviour is now implemented directly in geom_contour2
# and geom_contour_fill
ggplot(geopotential[date == date[1]], aes(lon, lat)) +
    geom_contour2(aes(z = gh), xwrap = c(-180, 360 + 180))
```

# Index