

# Package ‘metR’

November 25, 2020

**Type** Package

**Language** en-GB

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

**Version** 0.9.0

**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, lubridate, memoise, plyr, scales, sp, stringr, purrr, RCurl, isoband

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

**ByteCompile** yes

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 7.1.1

**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](mailto:elio.campitelli@cima.fcen.uba.ar)>

**Repository** CRAN

**Date/Publication** 2020-11-25 12:20:02 UTC

**R topics documented:**

Anomaly	3
as.path	4
ConvertLongitude	4
coriolis	5
cut.eof	6
denormalise	6
Derivate	7
DivideTimeseries	9
EOF	10
EPflux	12
FitLm	13
geom_arrow	14
geom_contour2	18
geom_contour_fill	21
geom_contour_tanaka	24
geom_label_contour	26
geom_relief	30
geom_streamline	32
geopotential	37
GeostrophicWind	37
GetSMNData	38
GetTopography	40
guide_vector	41
Impute2D	43
ImputeEOF	43
Interpolate	45
is.cross	47
JumpBy	48
label_placement_fraction	49
logic	49
Mag	50
MakeBreaks	51
map_labels	52
MaskLand	53
metR	54
Percentile	55
ReadNetCDF	56
reverselog_trans	58
scale_divergent	59
scale_fill_discretised	62
scale_longitude	67
scale_mag	69
season	70
spherical	71
stat_na	72
stat_subset	74

<i>Anomaly</i>	3
surface . . . . .	75
temperature . . . . .	76
thermodynamics . . . . .	76
Trajectory . . . . .	78
WaveFlux . . . . .	79
waves . . . . .	80
WrapCircular . . . . .	82
<b>Index</b>	<b>84</b>

---

Anomaly	<i>Anomalies</i>
---------	------------------

---

### 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 <a href="#">mean</a> such as <code>na.rm</code>

### 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.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, it will be recycled.

n number of points to interpolate to

path either TRUE or 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	<i>Remove some principal components.</i>
---------	--

---

**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	<i>Denormalise eof matrices</i>
-------------	---------------------------------

---

**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  $v_x + v_y \sim x + y$  (**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

<code>g</code>	ggplot object
<code>x</code>	The vector that was used in <code>g</code> for the x axis (must be of class Date)
<code>n</code>	Number of panels
<code>xlab</code>	x axis label
<code>ylab</code>	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](#), [reverse\\_log\\_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
)
```

**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 $\leq 1$ .
probs	the probabilities of the lower and upper values of estimated confidence intervals. If named, it's names will be used as column names.

<code>rotate</code>	if TRUE, scores and loadings will be rotated using <a href="#">varimax</a>
<code>suffix</code>	character to name the principal components
<code>fill</code>	value to infill implicit missing values or NULL if the data is dense.

### 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 a 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. <http://doi.org/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 weighed 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)
screplot(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 F<sub>lon</sub>, F<sub>lat</sub> and F<sub>lev</sub> 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. [https://doi.org/10.1175/1520-0469\(1985\)042<0217:OTDPO>2.0.CO;2](https://doi.org/10.1175/1520-0469(1985)042<0217:OTDPO>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. <https://doi.org/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 = rep(1, length(y)), se = FALSE, r2 = se)
```

**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

**Value**

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)

If there’s no complete cases in the regression, NAs are returned with no warning.

**Examples**

```
# Linear trend with "significant" 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 <code>aes()</code> or <code>aes_()</code> . If specified and <code>inherit.aes = TRUE</code> (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: If <code>NULL</code> , the default, the data is inherited from the plot data as specified in the call to <code>ggplot()</code> . A <code>data.frame</code> , or other object, will override the plot data. All objects will be fortified to produce a data frame. See <code>fortify()</code> for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a <code>data.frame</code> , and will be used as the layer data. A function can be created from a formula (e.g. <code>~ head(.x, 10)</code> ).
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 <code>layer()</code> . These are often aesthetics, used to set an aesthetic to a fixed value, like <code>colour = "red"</code> or <code>size = 3</code> . 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 <code>grid::arrow</code>
arrow	specification for arrow heads, as created by <code>arrow()</code> .
lineend	Line end style (round, butt, square).
na.rm	If <code>FALSE</code> , the default, missing values are removed with a warning. If <code>TRUE</code> , missing values are silently removed.
show.legend	logical. Should this layer be included in the legends? <code>NA</code> , the default, includes if any aesthetics are mapped. <code>FALSE</code> never includes, and <code>TRUE</code> always includes. It can also be a named logical vector to finely select the aesthetics to display.
inherit.aes	If <code>FALSE</code> , 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. <code>borders()</code> .



## 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 longitude 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))

```

---

geom\_contour2

2d contours of a 3d surface

---

### Description

A copy of `ggplot2::geom_contour` that accepts a function as the breaks argument and makes gaps for labels 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,
    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 <code>aes()</code> or <code>aes_()</code> . If specified and <code>inherit.aes = TRUE</code> (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: If <code>NULL</code> , the default, the data is inherited from the plot data as specified in the call to <code>ggplot()</code> . A <code>data.frame</code> , or other object, will override the plot data. All objects will be fortified to produce a data frame. See <code>fortify()</code> for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a <code>data.frame</code> , and will be used as the layer data. A function can be created from a formula (e.g. <code>~ head(.x, 10)</code> ).
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 <code>layer()</code> . These are often aesthetics, used to set an aesthetic to a fixed value, like <code>colour = "red"</code> or <code>size = 3</code> . 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:

	<ul style="list-style-type: none"> <li>• A numeric vector of breaks</li> <li>• A function that takes the range of the data and binwidth as input and returns breaks as output</li> </ul>
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. <ul style="list-style-type: none"> <li>• FALSE for letting the computation fail with no interpolation</li> <li>• TRUE for imputing missing values with <a href="#">Impute2D</a></li> <li>• A numeric value for constant imputation</li> <li>• A function that takes a vector and returns a numeric (e.g. mean)</li> </ul>
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. <a href="#">borders()</a> .
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):

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

### 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)
ggplot(reshape2::melt(volcano), aes(Var1, Var2)) +
  geom_contour2(aes(z = value, color = ..level..),
               breaks = AnchorBreaks(130, binwidth = 11))
```

---

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 <code>aes()</code> or <code>aes_()</code> . If specified and <code>inherit.aes = TRUE</code> (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: If <code>NULL</code> , the default, the data is inherited from the plot data as specified in the call to <code>ggplot()</code> . A <code>data.frame</code> , or other object, will override the plot data. All objects will be fortified to produce a data frame. See <code>fortify()</code> for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a <code>data.frame</code> , and will be used as the layer data. A function can be created from a formula (e.g. <code>~ head(.x, 10)</code> ).
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 <code>layer()</code> . These are often aesthetics, used to set an aesthetic to a fixed value, like <code>colour = "red"</code> or <code>size = 3</code> . They may also be parameters to the paired <code>geom/stat</code> .
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. <ul style="list-style-type: none"> <li>• <code>FALSE</code> for letting the computation fail with no interpolation</li> <li>• <code>TRUE</code> for imputing missing values with <code>Impute2D</code></li> <li>• A numeric value for constant imputation</li> </ul>

	<ul style="list-style-type: none"> <li>• A function that takes a vector and returns a numeric (e.g. mean)</li> </ul>
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. <code>borders()</code> .
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 <code>aes()</code> or <code>aes_()</code> . If specified and <code>inherit.aes = TRUE</code> (the default), it is combined with the default mapping at the top level of the plot. You must supply <code>mapping</code> if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If <code>NULL</code> , the default, the data is inherited from the plot data as specified in the call to <code>ggplot()</code> . A <code>data.frame</code> , or other object, will override the plot data. All objects will be fortified to produce a data frame. See <code>fortify()</code> for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a <code>data.frame</code> , and will be used as the layer data. A function can be created from a formula (e.g. <code>~ head(.x, 10)</code> ).
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 <code>layer()</code> . These are often aesthetics, used to set an aesthetic to a fixed value, like <code>colour = "red"</code> or <code>size = 3</code> . They may also be parameters to the paired geom/stat.
breaks	One of: <ul style="list-style-type: none"> <li>• A numeric vector of breaks</li> <li>• A function that takes the range of the data and binwidth as input and returns breaks as output</li> </ul>
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. <code>borders()</code> .

## 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.placement = label_placement_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.placement = label_placement_flattest(),
  parse = FALSE,
  nudge_x = 0,
  nudge_y = 0,
  stroke = 0,
  stroke.color = "white",
  check_overlap = FALSE,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

```

### Arguments

mapping	Set of aesthetic mappings created by <a href="#">aes()</a> or <a href="#">aes_()</a> . If specified and <code>inherit.aes = TRUE</code> (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: If <code>NULL</code> , the default, the data is inherited from the plot data as specified in the call to <a href="#">ggplot()</a> . A <code>data.frame</code> , or other object, will override the plot data. All objects will be fortified to produce a data frame. See <a href="#">fortify()</a> for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a <code>data.frame</code> , and will be used as the layer data. A function can be created from a formula (e.g. <code>~ head(.x, 10)</code> ).
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 jointly specified with <code>nudge_x</code> or <code>nudge_y</code> .
...	Other arguments passed on to <code>layer()</code> . These are often aesthetics, used to set an aesthetic to a fixed value, like <code>colour = "red"</code> or <code>size = 3</code> . They may also be parameters to the paired <code>geom/stat</code> .
min.size	minimum number of points for a contour to be labelled.
skip	number of contours to skip
label.placement	A function for placing labels (see <code>label_placement_flattest()</code> ).
parse	If TRUE, the labels will be parsed into expressions and displayed as described in <code>?plotmath</code> .
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 <code>position</code> .
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 <code>position</code> .
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. <code>borders()</code> .
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.
stroke.color	any valid colour.
check_overlap	If TRUE, text that overlaps previous text in the same layer will not be plotted. <code>check_overlap</code> happens at draw time and in the order of the data. Therefore data should be arranged by the label column before calling <code>geom_label()</code> or <code>geom_text()</code> .

## 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
- 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), rotate = FALSE)

g + geom_text_contour(aes(z = value),
  label.placement = label_placement_random())

g + geom_text_contour(aes(z = value),
  label.placement = label_placement_n(3))

g + geom_text_contour(aes(z = value),
  label.placement = label_placement_flattest())

g + geom_text_contour(aes(z = value),
  label.placement = label_placement_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: If NULL, the default, the data is inherited from the plot data as specified in the call to <code>ggplot()</code> . A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See <code>fortify()</code> for which variables will be created. 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. <code>~ head(.x, 10)</code> ).
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 <code>layer()</code> . These are often aesthetics, used to set an aesthetic to a fixed value, like <code>colour = "red"</code> or <code>size = 3</code> . 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 <code>ggplot2::geom_raster</code> , if FALSE, uses <code>ggplot2::geom_tile</code> .
interpolate	If TRUE interpolate linearly, if FALSE (the default) don't interpolate.
shadow	if TRUE, adds also a layer of <code>geom_shadow()</code>
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. <code>borders()</code> .
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 <a href="#">aes()</a> or <a href="#">aes_()</a> . If specified and <code>inherit.aes = TRUE</code> (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: If <code>NULL</code> , the default, the data is inherited from the plot data as specified in the call to <a href="#">ggplot()</a> . A <code>data.frame</code> , or other object, will override the plot data. All objects will be fortified to produce a data frame. See <a href="#">fortify()</a> for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a <code>data.frame</code> , and will be used as the layer data. A function can be created from a formula (e.g. <code>~ head(.x, 10)</code> ).
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 <a href="#">layer()</a> . These are often aesthetics, used to set an aesthetic to a fixed value, like <code>colour = "red"</code> or <code>size = 3</code> . They may also be parameters to the paired <code>geom/stat</code> .
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 <code>skip</code> if not <code>NULL</code> )
jitter, jitter.x, jitter.y	amount of jitter of the starting points
arrow.angle	parameters passed to <a href="#">grid::arrow</a>
arrow.length	parameters passed to <a href="#">grid::arrow</a>
arrow.ends	parameters passed to <a href="#">grid::arrow</a>
arrow.type	parameters passed to <a href="#">grid::arrow</a>
arrow	specification for arrow heads, as created by <a href="#">arrow()</a> .
lineend	Line end style (round, butt, square).

<code>na.rm</code>	If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.
<code>show.legend</code>	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.
<code>inherit.aes</code>	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. <code>borders()</code> .
<code>geom</code>	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 or 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	<i>Geopotential height</i>
--------------	----------------------------

---

### 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	<i>Calculate geostrophic winds</i>
-----------------	------------------------------------

---

### 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()
```

**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**

<code>date</code>	date vector of dates to fetch data
<code>type</code>	type of data to retrieve
<code>bar</code>	logical object indicating whether to show a progress bar
<code>cache</code>	logical indicating if the results should be saved on disk
<code>file.dir</code>	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<sup>2</sup>  
**diffuse** diffuse radiation in W/m<sup>2</sup>  
**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

<code>title</code>	A character string or expression indicating a title of guide. If <code>NULL</code> , the title is not shown. By default ( <code>waiver()</code> ), the name of the scale object or the name specified in <code>labs()</code> is used for the title.
<code>title.position</code>	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."
<code>title.theme</code>	A theme object for rendering the title text. Usually the object of <code>element_text()</code> is expected. By default, the theme is specified by <code>legend.title</code> in <code>theme()</code> or <code>theme</code> .
<code>title.hjust</code>	A number specifying horizontal justification of the title text.
<code>title.vjust</code>	A number specifying vertical justification of the title text.
<code>label</code>	logical. If <code>TRUE</code> then the labels are drawn. If <code>FALSE</code> then the labels are invisible.
<code>label.position</code>	A character string indicating the position of a label. One of "top", "bottom" (default for horizontal guide), "left", or "right" (default for vertical guide).
<code>label.theme</code>	A theme object for rendering the label text. Usually the object of <code>element_text()</code> is expected. By default, the theme is specified by <code>legend.text</code> in <code>theme()</code> .
<code>label.hjust</code>	A numeric specifying horizontal justification of the label text.
<code>label.vjust</code>	A numeric specifying vertical justification of the label text.
<code>keywidth</code>	A numeric or a <code>grid::unit()</code> object specifying the width of the legend key. Default value is <code>legend.key.width</code> or <code>legend.key.size</code> in <code>theme()</code> .
<code>keyheight</code>	A numeric or a <code>grid::unit()</code> object specifying the height of the legend key. Default value is <code>legend.key.height</code> or <code>legend.key.size</code> in <code>theme()</code> .
<code>direction</code>	A character string indicating the direction of the guide. One of "horizontal" or "vertical."
<code>default.unit</code>	A character string indicating <code>grid::unit()</code> for <code>keywidth</code> and <code>keyheight</code> .
<code>override.aes</code>	A list specifying aesthetic parameters of legend key. See details and examples.
<code>nrow</code>	The desired number of rows of legends.
<code>ncol</code>	The desired number of column of legends.
<code>byrow</code>	logical. If <code>FALSE</code> (the default) the legend-matrix is filled by columns, otherwise the legend-matrix is filled by rows.
<code>reverse</code>	logical. If <code>TRUE</code> the order of legends is reversed.
<code>order</code>	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.
<code>...</code>	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 <a href="#">mean</a> )

**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, <https://doi.org/10.5194/os-2-183-2006>, 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 longitudes
path <- geopotential[, Interpolate(gh ~ lon + lat, as.path(lons, lats))]
```

---

is.cross	<i>Cross pattern</i>
----------	----------------------

---

## 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 density. 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
```



---

label\_placement\_fraction  
*Functions to place contour labels*

---

### Description

These functions compute the position of contour labels

### Usage

```
label_placement_fraction(frac = 0.5)
```

```
label_placement_n(n = 2)
```

```
label_placement_random(seed = 42, n = 1)
```

```
label_placement_all()
```

```
label_placement_flattest(n = 1, ref_angle = 0)
```

```
label_placement_minmax(direction = c("vertical", "horizontal"))
```

### Arguments

frac	A numeric vector with values between 0 and 1 representing where in the contour to put labels (i.e. frac = 0.5 puts labels at the midpoint).
n	Number of labels to put.
seed	Seed to use for randomly choosing where to put labels.
ref_angle	Angle (in degrees counter-clockwise from East) to try to approximate labels.
direction	Direction in which to compute the maximum and minimum.

---

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 of a vector*

---

**Description**

Computes the magnitude of a vector of any dimension.

**Usage**

```
Mag(...)
```

**Arguments**

...                numeric vectors of coordinates or list of coordinates

**Details**

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

**Value**

A numeric vector the same length as each element of ... that is  $\sqrt{(x^2 + y^2 + \dots)}$ .

**See Also**

Other utilities: [Anomaly\(\)](#), [JumpBy\(\)](#), [Percentile\(\)](#), [logic](#)

**Examples**

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

# There's no vector recycling!
## 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()
```

**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

<code>file</code>	source to read from. Must be one of: <ul style="list-style-type: none"> <li>• A string representing a local file with read access.</li> <li>• A string representing a URL readable by <a href="#">ncdf4::nc_open()</a>. (this includes DAP urls).</li> <li>• A <code>netcdf</code> object returned by <a href="#">ncdf4::nc_open()</a>.</li> </ul>
<code>vars</code>	a character vector with the name of the variables to read. If <code>NULL</code> , then it reads all the variables.
<code>out</code>	character indicating the type of output desired
<code>subset</code>	a list of subsetting objects. See below.
<code>key</code>	if <code>TRUE</code> , returns a <code>data.table</code> keyed by the dimensions of the data.
<code>...</code>	in <a href="#">GlanceNetCDF()</a> , ignored. Is there for convenience so that a call to <a href="#">ReadNetCDF()</a> can be also valid for <a href="#">GlanceNetCDF()</a> .

### 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 an 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 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.
# iMake sure it's on the same exact grid!
field[, var2 := ReadNetCDF(file, out = "vector")]

## Not run:
if (!interactive())
# 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-negligible
# 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"))

## 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")
```

---

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:

- 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\(\)](#))

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.

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

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. 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()`.

`oob` One of:

- Function that handles limits outside of the scale limits (out of bounds).
- 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(FALSE),
position = "left",
rescaler = scales::rescale,
oob = scales::squish,
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\(\)](#))

**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.

**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

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 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()`.

oob One of:

- Function that handles limits outside of the scale limits (out of bounds).
- 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 <code>aesthetics = c("colour", "fill")</code> .
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., <code>scales::area_pal()</code> ).
name	The name of the scale. Used as the axis or legend title. If <code>waiver()</code> , the default, the name of the scale is taken from the first mapping used for that aesthetic. If <code>NULL</code> , the legend title will be omitted.
breaks	One of: <ul style="list-style-type: none"> <li>• <code>NULL</code> for no breaks</li> <li>• <code>waiver()</code> for the default breaks computed by the <a href="#">transformation object</a></li> <li>• A numeric vector of positions</li> <li>• A function that takes the limits as input and returns breaks as output (e.g., a function returned by <code>scales::extended_breaks()</code>)</li> </ul>
labels	One of: <ul style="list-style-type: none"> <li>• <code>NULL</code> for no labels</li> <li>• <code>waiver()</code> for the default labels computed by the transformation object</li> <li>• A character vector giving labels (must be same length as breaks)</li> <li>• A function that takes the breaks as input and returns labels as output</li> </ul>
limits	One of: <ul style="list-style-type: none"> <li>• <code>NULL</code> to use the default scale range</li> <li>• A numeric vector of length two providing limits of the scale. Use <code>NA</code> to refer to the existing minimum or maximum</li> <li>• A function that accepts the existing (automatic) limits and returns new limits Note that setting limits on positional scales will <b>remove</b> data outside of the limits. If the purpose is to zoom, use the <code>limit</code> argument in the coordinate system (see <code>coord_cartesian()</code>).</li> </ul>
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 <code>&lt;name&gt;_trans</code> (e.g., <code>scales::boxcox_trans()</code> ). You can create your own transformation with <code>scales::trans_new()</code> .
drop	Should unused factor levels be omitted from the scale? The default, <code>TRUE</code> , uses the levels that appear in the data; <code>FALSE</code> 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 <code>scales::rescale()</code> , except for diverging and n colour gradients (i.e., <code>scale_colour_gradient2()</code> , <code>scale_colour_gradientn()</code> ). The rescaler is ignored by position scales, which always use <code>scales::rescale()</code> .
oob	One of: <ul style="list-style-type: none"> <li>• Function that handles limits outside of the scale limits (out of bounds).</li> <li>• The default (<code>scales::censor()</code>) replaces out of bounds values with NA.</li> <li>• <code>scales::squish()</code> for squishing out of bounds values into range.</li> <li>• <code>scales::squish_infinite()</code> for squishing infinite values into range.</li> </ul>
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 on 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 underlying 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")

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, guide = guide_colorsteps())

# 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\_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 = 60,  
  breaks = seq(-90, 90, by = ticks),  
  expand = c(0, 0),  
  labels = LatLabel,  
  trans = "identity",  
  ...  
)
```

```

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

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

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

```

## Arguments

name	The name of the scale. Used as the axis or legend title. If <code>waiver()</code> , the default, the name of the scale is taken from the first mapping used for that aesthetic. If <code>NULL</code> , the legend title will be omitted.
ticks	spacing between breaks
breaks	One of: <ul style="list-style-type: none"> <li>• <code>NULL</code> for no breaks</li> <li>• <code>waiver()</code> for the default breaks computed by the <a href="#">transformation object</a></li> <li>• A numeric vector of positions</li> <li>• A function that takes the limits as input and returns breaks as output (e.g., a function returned by <code>scales::extended_breaks()</code>)</li> </ul>
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 <code>expansion()</code> to generate the values for the <code>expand</code> 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: <ul style="list-style-type: none"> <li>• <code>NULL</code> for no labels</li> <li>• <code>waiver()</code> for the default labels computed by the transformation object</li> <li>• A character vector giving labels (must be same length as breaks)</li> <li>• A function that takes the breaks as input and returns labels as output</li> </ul>
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 <code>&lt;name&gt;_trans</code> (e.g., <code>scales::boxcox_trans()</code> ). You can create your own transformation with <code>scales::trans_new()</code> .
...	Other arguments passed on to <code>scale_(xly)_continuous()</code>

**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()
```

---

 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 <code>waiver()</code> , the default, the name of the scale is taken from the first mapping used for that aesthetic. If <code>NULL</code> , the legend title will be omitted.
labels	One of: <ul style="list-style-type: none"> <li>• <code>NULL</code> for no labels</li> <li>• <code>waiver()</code> for the default labels computed by the transformation object</li> <li>• A character vector giving labels (must be same length as breaks)</li> <li>• A function that takes the breaks as input and returns labels as output</li> </ul>
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 <code>waiver()</code>
guide	type of legend
...	Other arguments passed on to <code>scale_(xly)_continuous()</code>

**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	<i>Assign seasons to months</i>
--------	---------------------------------

---

**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. `seasonally()` 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	<i>Filter only NA values.</i>
---------	-------------------------------

---

## 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 <code>aes()</code> or <code>aes_()</code> . If specified and <code>inherit.aes = TRUE</code> (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: If <code>NULL</code> , the default, the data is inherited from the plot data as specified in the call to <code>ggplot()</code> . A <code>data.frame</code> , or other object, will override the plot data. All objects will be fortified to produce a data frame. See <code>fortify()</code> for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a <code>data.frame</code> , and will be used as the layer data. A function can be created from a formula (e.g. <code>~ head(.x, 10)</code> ).
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 <code>layer()</code> . These are often aesthetics, used to set an aesthetic to a fixed value, like <code>colour = "red"</code> or <code>size = 3</code> . 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. <code>borders()</code> .

### 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	<i>Subset values</i>
-------------	----------------------

---

### 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 <code>aes()</code> or <code>aes_()</code> . If specified and <code>inherit.aes = TRUE</code> (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: If NULL, the default, the data is inherited from the plot data as specified in the call to <code>ggplot()</code> . A <code>data.frame</code> , or other object, will override the plot data. All objects will be fortified to produce a data frame. See <code>fortify()</code> for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a <code>data.frame</code> , and will be used as the layer data. A function can be created from a formula (e.g. <code>~ head(.x, 10)</code> ).
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 <code>layer()</code> . These are often aesthetics, used to set an aesthetic to a fixed value, like <code>colour = "red"</code> or <code>size = 3</code> . 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. <code>borders()</code> .

**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	<i>Air temperature</i>
-------------	------------------------

---

**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	<i>Thermodynamics</i>
----------------	-----------------------

---

**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  $\theta = T(P_0/P)^\kappa$ .

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  $kg/m^3$ . ClausiusClapeyron 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](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 <- Adiabats(70000, 20 + 273.15))
Adiabats(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 $dx + dy \sim x + y + t$ .
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, [https://journals.ametsoc.org/doi/abs/10.1175/1520-0469\(2001\)058%3C0608%3AAFOAPI%3E2.0.CO%3B2](https://journals.ametsoc.org/doi/abs/10.1175/1520-0469(2001)058%3C0608%3AAFOAPI%3E2.0.CO%3B2)

Adapted from [https://github.com/marisolosman/Reunion\\_Clima/blob/master/WAF/Calculo\\_WAF.ipynb](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:

$$y = \sum A \cos((x - \phi)k)$$

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 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, [https://doi.org/10.1175/1520-0493\(2003\)131<1011:EEORWP>2.0.CO;2](https://doi.org/10.1175/1520-0493(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))

# Additionally 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

- \* **datasets**
  - geom\_arrow, 14
  - geom\_contour2, 18
  - geom\_contour\_fill, 21
  - geom\_contour\_tanaka, 24
  - geom\_label\_contour, 26
  - geom\_relief, 30
  - geom\_streamline, 32
  - geopotential, 37
  - scale\_fill\_discretised, 62
  - stat\_na, 72
  - stat\_subset, 74
  - surface, 75
  - temperature, 76
- \* **ggplot2 helpers**
  - DivideTimeseries, 9
  - geom\_arrow, 14
  - geom\_contour2, 18
  - geom\_contour\_fill, 21
  - geom\_label\_contour, 26
  - geom\_relief, 30
  - geom\_streamline, 32
  - MakeBreaks, 51
  - map\_labels, 52
  - reverselog\_trans, 58
  - scale\_divergent, 59
  - scale\_longitude, 67
  - stat\_na, 72
  - stat\_subset, 74
  - WrapCircular, 82
- \* **meteorology functions**
  - Derivate, 7
  - EOF, 10
  - GeostrophicWind, 37
  - thermodynamics, 76
  - WaveFlux, 79
  - waves, 80
- \* **utilities**
  - Anomaly, 3
  - JumpBy, 48
  - logic, 49
  - Mag, 50
  - Percentile, 55
  - %~%(logic), 49
  - %in%, 50
  - Adiabat (thermodynamics), 76
  - aes(), 16, 19, 22, 24, 27, 30, 34, 72, 74
  - aes\_(), 16, 19, 22, 24, 27, 30, 34, 72, 74
  - AnchorBreaks (MakeBreaks), 51
  - Anomaly, 3, 48, 50, 51, 55
  - as.path, 4, 46
  - AssignSeason (season), 70
  - autoplot, 11
  - base::svd, 11
  - borders(), 16, 20, 23, 25, 28, 31, 35, 73, 74
  - BuildWave (waves), 80
  - ClausiusClapeyron (thermodynamics), 76
  - continuous\_scale, 60, 63
  - ConvertLongitude, 4
  - coord\_cartesian(), 61, 64, 65
  - coriolis, 5
  - cross (is.cross), 47
  - cut.eof, 6, 11
  - data.table, 55
  - data.table::dcast, 44
  - denormalise, 6
  - denormalize (denormalise), 6
  - Derivate, 7, 11, 38, 54, 78, 79, 81
  - DewPoint (thermodynamics), 76
  - discretised\_scale
    - (scale\_fill\_discretised), 62
  - Divergence (Derivate), 7
  - DivideTimeseries, 9, 17, 21, 23, 29, 32, 36, 52, 53, 59, 62, 69, 73, 75, 83
  - dlat (spherical), 71

- dlon (spherical), 71
- dx (spherical), 71
- dy (spherical), 71
- element\_text(), 42
- EOF, 8, 10, 38, 55, 78, 79, 81
- EOF(), 6
- EPflux, 12
- expansion(), 61, 64, 68
- f (coriolis), 5
- fft, 80
- fields::interp.surface, 45
- FilterWave (waves), 80
- FitLm, 13
- FitWave (waves), 80
- Formula::Formula, 11, 44
- fortify(), 16, 19, 22, 24, 27, 31, 34, 72, 74
- geom\_arrow, 10, 14, 21, 23, 29, 32, 36, 52, 53, 59, 62, 69, 73, 75, 83
- geom\_contour, 21
- geom\_contour2, 10, 17, 18, 23, 29, 32, 36, 52, 53, 59, 62, 69, 73, 75, 83
- geom\_contour\_fill, 10, 17, 21, 21, 29, 32, 36, 51–53, 59, 62, 69, 73, 75, 83
- geom\_contour\_fill(), 43, 66
- geom\_contour\_tanaka, 24
- geom\_label\_contour, 10, 17, 21, 23, 26, 32, 36, 52, 53, 59, 62, 69, 73, 75, 83
- geom\_relief, 10, 17, 21, 23, 29, 30, 36, 52, 53, 59, 62, 69, 73, 75, 83
- geom\_shadow (geom\_relief), 30
- geom\_streamline, 10, 17, 21, 23, 29, 32, 32, 52, 53, 59, 62, 69, 73, 75, 83
- geom\_text\_contour (geom\_label\_contour), 26
- geom\_vector (geom\_arrow), 14
- GeomArrow (geom\_arrow), 14
- GeomContour2 (geom\_contour2), 18
- GeomContourTanaka (geom\_contour\_tanaka), 24
- GeomLabelContour (geom\_label\_contour), 26
- GeomRelief (geom\_relief), 30
- GeomShadow (geom\_relief), 30
- GeomStreamline (geom\_streamline), 32
- GeomTextContour (geom\_label\_contour), 26
- geopotential, 37
- GeostrophicWind, 8, 11, 37, 78, 79, 81
- GetSMNData, 38
- GetTopography, 40
- ggplot(), 16, 19, 22, 24, 27, 31, 34, 72, 74
- ggplot2, 54
- ggplot2::binned\_scale(), 62
- ggplot2::geom\_contour, 18, 21
- ggplot2::geom\_raster, 31
- ggplot2::geom\_segment, 14
- ggplot2::geom\_tile, 31
- ggplot2::stat\_contour, 26, 28, 52
- GlanceNetCDF (ReadNetCDF), 56
- GlanceNetCDF(), 56
- grid::arrow, 16, 34
- grid::unit(), 42
- guide\_colourstrip, 10, 17, 21, 23, 29, 32, 36, 52, 53, 59, 62, 69, 73, 75, 83
- guide\_vector, 41
- IdealGas (thermodynamics), 76
- Impute2D, 20, 22, 43
- ImputeEOF, 43
- Interpolate, 4, 45
- irlba::irlba, 11
- is.cross, 47
- is.full\_season (season), 70
- JumpBy, 3, 48, 50, 51, 55
- label\_placement\_all (label\_placement\_fraction), 49
- label\_placement flattest (label\_placement\_fraction), 49
- label\_placement\_fraction, 49
- label\_placement\_minmax (label\_placement\_fraction), 49
- label\_placement\_n (label\_placement\_fraction), 49
- label\_placement\_random (label\_placement\_fraction), 49
- labs(), 42
- Laplacian (Derivate), 7
- LatLabel (map\_labels), 52
- layer(), 16, 19, 22, 25, 28, 31, 34, 73, 74
- logic, 3, 48, 49, 51, 55
- LonLabel (map\_labels), 52
- Mag, 3, 48, 50, 50, 55
- MakeBreaks, 10, 17, 21, 23, 29, 32, 36, 51, 53, 59, 62, 69, 73, 75, 83

- map, 53
- map\_labels, 10, 17, 21, 23, 29, 32, 36, 52, 52, 59, 62, 69, 73, 75, 83
- MaskLand, 53
- mean, 3, 43
- metR, 54
- metR-package (metR), 54
- MixingRatio (thermodynamics), 76
  
- ncdf4::nc\_open(), 56
- ncdf4::ncvar\_get, 56
- ncvar\_get, 56
  
- Percentile, 3, 48, 50, 51, 55
- predict, 11
  
- ReadNetCDF, 56
- ReadNetCDF(), 56
- RepeatCircular (WrapCircular), 82
- reverselog\_trans, 10, 17, 21, 23, 29, 32, 36, 52, 53, 58, 62, 69, 73, 75, 83
  
- scale\_color\_divergent (scale\_divergent), 59
- scale\_colour\_divergent (scale\_divergent), 59
- scale\_colour\_gradient2, 59
- scale\_colour\_gradient2(), 61, 64, 66
- scale\_colour\_gradientn(), 61, 64, 66
- scale\_divergent, 10, 17, 21, 23, 29, 32, 36, 52, 53, 59, 59, 69, 73, 75, 83
- scale\_fill\_discretised, 62
- scale\_fill\_discretised(), 23
- scale\_fill\_divergent (scale\_divergent), 59
- scale\_fill\_divergent\_discretised (scale\_fill\_discretised), 62
- scale\_latitude (scale\_longitude), 67
- scale\_longitude, 10, 17, 21, 23, 29, 32, 36, 52, 53, 59, 62, 67, 73, 75, 83
- scale\_mag, 69
- scale\_mag\_continuous (scale\_mag), 69
- scale\_x\_continuous, 67
- scale\_x\_latitude (scale\_longitude), 67
- scale\_x\_level (scale\_longitude), 67
- scale\_x\_longitude (scale\_longitude), 67
- scale\_y\_continuous, 67
- scale\_y\_latitude (scale\_longitude), 67
- scale\_y\_level, 54
- scale\_y\_level (scale\_longitude), 67
- scale\_y\_longitude (scale\_longitude), 67
- ScaleDiscretised (scale\_fill\_discretised), 62
- scales::area\_pal(), 60, 63, 65
- scales::boxcox\_trans(), 61, 64, 65, 68
- scales::censor(), 61, 64, 66
- scales::extended\_breaks(), 60, 63, 65, 68
- scales::rescale(), 61, 64, 66
- scales::squish(), 61, 64, 66
- scales::squish\_infinite(), 61, 64, 66
- scales::trans\_new(), 61, 64, 65, 68
- screepplot, 11
- season, 70
- seasonally (season), 70
- Similar (logic), 49
- spherical, 35, 71
- stat\_contour2 (geom\_contour2), 18
- stat\_contour\_fill, 54
- stat\_contour\_fill (geom\_contour\_fill), 21
- stat\_na, 10, 17, 21, 23, 29, 32, 36, 52, 53, 59, 62, 69, 72, 75, 83
- stat\_streamline (geom\_streamline), 32
- stat\_subset, 10, 17, 21, 23, 29, 32, 36, 52, 53, 59, 62, 69, 73, 74, 83
- StatArrow (geom\_arrow), 14
- StatContour2 (geom\_contour2), 18
- StatContourFill (geom\_contour\_fill), 21
- StatNa (stat\_na), 72
- stats::lm.fit, 13
- StatStreamline (geom\_streamline), 32
- StatSubset (stat\_subset), 74
- StatTextContour (geom\_label\_contour), 26
- summary, 11
- surface, 75
  
- temperature, 76
- theme(), 42
- thermodynamics, 8, 11, 38, 76, 79, 81
- Trajectory, 78
- transformation object, 60, 63, 65, 68
  
- varimax, 11
- VirtualTemperature (thermodynamics), 76
- Vorticity (Derivate), 7
  
- waiver(), 42
- WaveEnvelope (waves), 80

WaveFlux, [8](#), [11](#), [38](#), [78](#), [79](#), [81](#)

waves, [8](#), [11](#), [38](#), [78](#), [79](#), [80](#)

WrapCircular, [10](#), [17](#), [21](#), [23](#), [29](#), [32](#), [36](#), [52](#),  
[53](#), [59](#), [62](#), [69](#), [73](#), [75](#), [82](#)