Package ‘metR’

April 10, 2020

Type Package
Language en-GB
Title Tools for Easier Analysis of Meteorological Fields
Version 0.7.0
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
Suggests maps, maptools, covr, irlba, knitr, ncd4, pkgdown, reshape2, rmarkdown, testthat (>= 2.1.0), viridis, udunits2, gridExtra, vdiffr
ByteCompile yes
Encoding UTF-8
LazyData true
RoxygenNote 7.1.0
VignetteBuilder knitr
NeedsCompilation no
Author Elio Campitelli [cre, aut] (<https://orcid.org/0000-0002-7742-9230>)
Maintainer Elio Campitelli <elio.campitelli@cima.fcen.uba.ar>
Repository CRAN
Date/Publication 2020-04-10 07:10:06 UTC
### R topics documented:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anomaly</td>
<td>3</td>
</tr>
<tr>
<td>as.path</td>
<td>4</td>
</tr>
<tr>
<td>ConvertLongitude</td>
<td>4</td>
</tr>
<tr>
<td>coriolis</td>
<td>5</td>
</tr>
<tr>
<td>cut.eof</td>
<td>6</td>
</tr>
<tr>
<td>denormalise</td>
<td>6</td>
</tr>
<tr>
<td>Derivate</td>
<td>7</td>
</tr>
<tr>
<td>DivideTimeseries</td>
<td>9</td>
</tr>
<tr>
<td>EOF</td>
<td>10</td>
</tr>
<tr>
<td>EPflux</td>
<td>12</td>
</tr>
<tr>
<td>FitLm</td>
<td>13</td>
</tr>
<tr>
<td>geom.arrow</td>
<td>14</td>
</tr>
<tr>
<td>geom_contour2</td>
<td>18</td>
</tr>
<tr>
<td>geom_contour_fill</td>
<td>21</td>
</tr>
<tr>
<td>geom_contour_tanaka</td>
<td>23</td>
</tr>
<tr>
<td>geom_label_contour</td>
<td>26</td>
</tr>
<tr>
<td>geom_relief</td>
<td>29</td>
</tr>
<tr>
<td>geom_streamline</td>
<td>31</td>
</tr>
<tr>
<td>geopotential</td>
<td>36</td>
</tr>
<tr>
<td>GeostrophicWind</td>
<td>37</td>
</tr>
<tr>
<td>GetSMNData</td>
<td>38</td>
</tr>
<tr>
<td>GetTopography</td>
<td>39</td>
</tr>
<tr>
<td>guide_vector</td>
<td>40</td>
</tr>
<tr>
<td>Impute2D</td>
<td>42</td>
</tr>
<tr>
<td>ImputeEOF</td>
<td>43</td>
</tr>
<tr>
<td>Interpolate</td>
<td>44</td>
</tr>
<tr>
<td>is.cross</td>
<td>46</td>
</tr>
<tr>
<td>JumpBy</td>
<td>47</td>
</tr>
<tr>
<td>logic</td>
<td>48</td>
</tr>
<tr>
<td>Mag</td>
<td>49</td>
</tr>
<tr>
<td>MakeBreaks</td>
<td>50</td>
</tr>
<tr>
<td>map_labels</td>
<td>51</td>
</tr>
<tr>
<td>MaskLand</td>
<td>52</td>
</tr>
<tr>
<td>metR</td>
<td>53</td>
</tr>
<tr>
<td>Percentile</td>
<td>54</td>
</tr>
<tr>
<td>ReadNetCDF</td>
<td>54</td>
</tr>
<tr>
<td>reverselog_trans</td>
<td>57</td>
</tr>
<tr>
<td>scale_divergent</td>
<td>58</td>
</tr>
<tr>
<td>scale_longitude</td>
<td>61</td>
</tr>
<tr>
<td>scale_mag</td>
<td>63</td>
</tr>
<tr>
<td>season</td>
<td>64</td>
</tr>
<tr>
<td>spherical</td>
<td>65</td>
</tr>
<tr>
<td>stat_na</td>
<td>66</td>
</tr>
<tr>
<td>stat_subset</td>
<td>68</td>
</tr>
<tr>
<td>temperature</td>
<td>69</td>
</tr>
<tr>
<td>thermodynamics</td>
<td>70</td>
</tr>
</tbody>
</table>
Description

Saves keystrokes for computing anomalies.

Usage

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

Arguments

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

Value

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

See Also

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

Examples

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

*Interpolates between locations*

**Description**

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

**Usage**

```r
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**

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

denormalise

Description

The matrices returned by \texttt{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,
Derivate

  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 \( \text{sphere} = \text{TRUE} \), 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 \( \text{fill} = \text{TRUE} \) will degrade the solution near the edges of a non-cyclical boundary. Use with caution.

\( \text{Laplacian()} \), \( \text{Divergence()} \) and \( \text{Vorticity()} \) are convenient wrappers that call \( \text{Derivate()} \) and make the appropriate sums. For \( \text{Divergence()} \) and \( \text{Vorticity()} \), \( \text{formula} \) must be of the form \( vx + vy \sim x + y \) (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: \( \text{EOF()}, \text{GeostrophicWind()}, \text{WaveFlux()}, \text{thermodynamics}, \text{waves} \)

Examples

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

  plot(theta, dx_analytical, type = "l")
  points(theta, dx_finite diff, col = "red")
DivideTimeseries

# 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  Divide long timeseries for better reading

Description

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

Usage

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

Arguments

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

Value

Draws a plot.
See Also

Other ggplot2 helpers: MakeBreaks(), WrapCircular(), geom_arrow(), geom_contour2(), geom_contour_fill(), geom_label_contour(), geom_relied(), geom_streamline(), guide_colourstrip(), map_labels, reversecolor_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")

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 <= 1.
probs the probabilities of the lower and upper values of estimated confidence intervals. If named, it’s names will be used as column names.
rotate if TRUE, scores and loadings will be rotated using varimax
suffix character to name the principal components
fill value to infill implicit missing values or NULL if the data is dense.

Details

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

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

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

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

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

Value

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

- **left** `data.table` with left singular vectors
- **right** `data.table` with right singular vectors
- **sdev** `data.table` with singular values, their explained variance, and, optionally, quantiles estimated via bootstrap

There are some methods implemented

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

References


See Also

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

# The Antarctic Oscillation is computed from the
# monthly geopotential height anomalies weighted 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)
screepplot(eof)

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

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

# AAO signal
geom$aao$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.
geom$left == 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 Computations Eliassen-Palm fluxes.

Description
Computes Eliassen-Palm fluxes.
### 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)
```

**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

---

**EPflux**

**Usage**

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

**Arguments**

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

**Value**

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

**References**


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

```r
# 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

```r
df <- data.frame(x = 1:4, y = 10:9)

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, unit(arrow.length, "lines"), ends = arrow.ends, type = arrow.type),
  lineend = "butt",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

df2 <- 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, unit(arrow.length, "lines"), ends = arrow.ends, type = arrow.type),
  lineend = "butt",
)
geomArrow

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 ggplot().
A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created.
A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x,10)).

stat     The statistical transformation to use on the data for this layer, as a string.

position Position adjustment, either as a string, or the result of a call to a position adjustment function.

...      Other arguments passed on to layer(). These are often aesthetics, used to set an aesthetic to a fixed value, like colour = "red" or size = 3. They may also be parameters to the paired geom/stat.

start    starting angle for rotation in degrees

direction direction of rotation (counter-clockwise or clockwise)

pivot    numeric indicating where to pivot the arrow where 0 means at the beginning and 1 means at the end.

preserve.dir logical indicating whether to preserve direction or not

min.mag   minimum magnitude for plotting vectors

skip, skip.x, skip.y numeric specifying number of gridpoints not to draw in the x and y direction

arrow.length, arrow.angle, arrow.ends, arrow.type parameters passed to grid::arrow

arrow    specification for arrow heads, as created by arrow().

lineend  Line end style (round, butt, square).

na.rm    If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.

show.legend logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display.

inherit.aes If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn’t inherit behaviour from the default plot specification, e.g. borders().

**Details**

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

**Aesthetics**

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

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

**See Also**

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

**Examples**

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

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

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

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

# A dramatic illustration of the difference between arrow and vector
g + coord_polar()
```
# When plotting winds in a lat-lon grid, a good way to have both
# the correct direction and an interpretable magnitude is to define
# the angle by the longitud and latitude displacement and the magnitude
# by the wind velocity. That way arrows are always parallel to streamlines
# and their magnitude are in the correct units.

```r
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

```r
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**

```r
geom_contour2(
  mapping = NULL,
  data = NULL,
  stat = "contour2",
  position = "identity",
  ...
)
```
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 `ggplot()`.
  - A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify()` for which variables will be created.
  - A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. `~ head(.x,10)`).

- **stat**
  The statistical transformation to use on the data for this layer, as a string.

- **position**
  Position adjustment, either as a string, or the result of a call to a position adjustment function.

- **...**
  Other arguments passed on to `layer()`. These are often aesthetics, used to set an aesthetic to a fixed value, like `colour = "red"` or `size = 3`. They may also be parameters to the paired geom/stat.

- **lineend**
  Line end style (round, butt, square).

- **linejoin**
  Line join style (round, mitre, bevel).

- **linemitre**
  Line mitre limit (number greater than 1).

- **breaks**
  One of:
  - A numeric vector of breaks
• A function that takes the range of the data and binwidth as input and returns breaks as output

bins Number of evenly spaced breaks.
binwidth Distance between breaks.
global.breaks Logical indicating whether breaks should be computed for the whole data or for each grouping.
na.rm If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.
na.fill How to fill missing values.
  • FALSE for letting the computation fail with no interpolation
  • TRUE for imputing missing values with Impute2D
  • A numeric value for constant imputation
  • A function that takes a vector and returns a numeric (e.g. mean)
show.legend logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display.
inherit.aes If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn’t inherit behaviour from the default plot specification, e.g. borders().
geom The geometric object to use display the data

Aesthetics

geom_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, reverse_log_trans(), scale_divergent, scale_longitude, stat_na(), stat_subset()
Examples

```r
library(ggplot2)
ggplot(reshape2::melt(volcano), aes(Var1, Var2)) +
  geom_contour2(aes(z = value, color = ..level..),
                breaks = AnchorBreaks(130, binwidth = 11))
```

---

**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**

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

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

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

data
The data to be displayed in this layer. There are three options:

If NULL, the default, the data is inherited from the plot data as specified in the call to `ggplot()`.

A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify()` for which variables will be created.

A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. `~ head(.x,10)`).

stat
The statistical transformation to use on the data for this layer, as a string.

position
Position adjustment, either as a string, or the result of a call to a position adjustment function.

... Other arguments passed on to `layer()`. These are often aesthetics, used to set an aesthetic to a fixed value, like `colour = "red"` or `size = 3`. They may also be parameters to the paired geom/stat.

breaks numeric vector of breaks

bins Number of evenly spaced breaks.

binwidth Distance between breaks.

na.fill How to fill missing values.

  • FALSE for letting the computation fail with no interpolation
  • TRUE for imputing missing values with `Impute2D`
  • A numeric value for constant imputation
  • A function that takes a vector and returns a numeric (e.g. mean)

show.legend logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes. It can also be a named logical vector to finely select the aesthetics to display.

inherit.aes If `FALSE`, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn’t inherit behaviour from the default plot specification, e.g. `borders()`.

geom The geometric object to use display the data

global.breaks Logical indicating whether breaks should be computed for the whole data or for each grouping.

Aesthetics

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

  • `x`
  • `y`
• alpha
• colour
• group
• linetype
• size
• weight

Computed variables

int.level value of the interior contour

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)

# If one uses level instead of int.level, one of the small
# contours near the crater disapears
ggplot(surface, aes(Var1, Var2, z = value)) +
  geom_contour_fill(aes(fill = ..level..))

Illuminated contours

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

```r
gem_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),
  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 `ggplot()`.
  - A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify()` for which variables will be created.
  - A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. `~ head(.x,10)`).

- **stat**: The statistical transformation to use on the data for this layer, as a string.

- **position**: Position adjustment, either as a string, or the result of a call to a position adjustment function.

- **...**: Other arguments passed on to `layer()`. These are often aesthetics, used to set an aesthetic to a fixed value, like `colour = "red"` or `size = 3`. They may also be parameters to the paired geom/stat.

- **breaks**: One of:
  - A numeric vector of breaks
  - A function that takes the range of the data and binwidth as input and returns breaks as output

- **bins**: Number of evenly spaced breaks.

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

Aesthetics

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

• x
• y
• z
• linetype

Examples

library(ggplot2)
library(data.table)
# A fresh look at the boring old volcano dataset
ggplot(reshape2::melt(volcano), aes(Var1, Var2)) +
  geom_contour_fill(aes(z = value)) +
  geom_contour_tanaka(aes(z = value)) +
  theme_void() +
  viridis::scale_fill_viridis(guide = "none")

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))

_____________

<table>
<thead>
<tr>
<th>geom_label_contour</th>
<th>Label contours</th>
</tr>
</thead>
</table>

**Description**

Draws labels on contours built with `ggplot2::stat_contour`.

**Usage**

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

```r
geom_text_contour(
  mapping = NULL,
  data = NULL,
  stat = "text_contour",
  position = "identity",
  ...,
  min.size = 5,
  skip = 0,
  rotate = TRUE,
  parse = FALSE,
```
geom_label_contour

```r
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 `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 `ggplot()`.
    - A `data.frame`, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify()` for which variables will be created.
    - A function will be called with a single argument, the plot data. The return value must be a `data.frame`, and will be used as the layer data. A function can be created from a `formula` (e.g. `~ head(.x,10)`).

- **stat**
  - The statistical transformation to use on the data for this layer, as a string.

- **position**
  - Position adjustment, either as a string, or the result of a call to a position adjustment function.

- **...**
  - Other arguments passed on to `layer()`. These are often aesthetics, used to set an aesthetic to a fixed value, like `colour = "red"` or `size = 3`. They may also be parameters to the paired geom/stat.

- **min.size**
  - Minimum number of points for a contour to be labelled.

- **skip**
  - Number of contours to skip.

- **parse**
  - If `TRUE`, the labels will be parsed into expressions and displayed as described in `plotmath`.

- **nudge_x**
  - Horizontal and vertical adjustment to nudge labels by. Useful for offsetting text from points, particularly on discrete scales.

- **nudge_y**
  - Horizontal and vertical adjustment to nudge labels by. Useful for offsetting text from points, particularly on discrete scales.

- **label.padding**
  - Amount of padding around label. Defaults to 0.25 lines.

- **label.r**
  - Radius of rounded corners. Defaults to 0.15 lines.

- **label.size**
  - Size of label border, in mm.

- **na.rm**
  - If `FALSE`, the default, missing values are removed with a warning. If `TRUE`, missing values are silently removed.
show.legend logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display.

inherit.aes If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn’t inherit behaviour from the default plot specification, e.g. borders().

rotate logical indicating whether to rotate text following the contour.

stroke numerical indicating width of stroke relative to the size of the text. Ignored if less than zero.

stroke.color any valid colour.

check_overlap If TRUE, text that overlaps previous text in the same layer will not be plotted.

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

```r
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)
```

---

**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**

```r
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
)
```

```r
geom_shadow(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  sun.angle = 60,
  range = c(0, 1),
  skip = 0,
  raster = TRUE,
  interpolate = 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 ggplot().
A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created.
A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x,10)).

stat The statistical transformation to use on the data for this layer, as a string.

position Position adjustment, either as a string, or the result of a call to a position adjustment function.

... Other arguments passed on to layer(). These are often aesthetics, used to set an aesthetic to a fixed value, like colour = "red" or size = 3. They may also be parameters to the paired geom/stat.

sun.angle angle from which the sun is shining, in degrees counterclockwise from 12 o’clock

raster if TRUE (the default), uses ggplot2::geom_raster, if FALSE, uses ggplot2::geom_tile.

interpolate If TRUE interpolate linearly, if FALSE (the default) don’t interpolate.

shadow if TRUE, adds also a layer of geom_shadow()

na.rm If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.

show.legend logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display.

inherit.aes If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn’t inherit behaviour from the default plot specification, e.g. borders().

range transparency range for shadows

skip data points to skip when casting shadows

Details

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

gem_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

gem_streamline(
  mapping = NULL,
  data = NULL,
  stat = "streamline",
  position = "identity",
  ...
  L = 5,
  min.L = 0,
  res = 1,
  S = NULL,
)
geom_streamline

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, 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,
geom_streamline

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

Arguments

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

stat The statistical transformation to use on the data for this layer, as a string.
position Position adjustment, either as a string, or the result of a call to a position adjustment function.

... Other arguments passed on to layer(). These are often aesthetics, used to set an aesthetic to a fixed value, like colour = "red" or size = 3. They may also be parameters to the paired geom/stat.

L, typical length of a streamline in x and y units
min.L minimum length of segments to show
res, resolution parameter (higher numbers increases the resolution)
S optional numeric number of timesteps for integration
dt optional numeric size "timestep" for integration
xwrap, ywrap vector of length two used to wrap the circular dimension.
skip numeric specifying number of gridpoints not to draw in the x and y direction
skip.x numeric specifying number of gridpoints not to draw in the x and y direction
skip.y numeric specifying number of gridpoints not to draw in the x and y direction
n, nx, ny optional numeric indicating the number of points to draw in the x and y direction (replaces skip if not NULL)
jitter, jitter.x, jitter.y amount of jitter of the starting points
arrow.angle parameters passed to grid::arrow
arrow.length: parameters passed to `grid::arrow`
arrow.ends: parameters passed to `grid::arrow`
arrows.type: parameters passed to `grid::arrow`
arrow: specification for arrow heads, as created by `arrow()`.
lineend: Line end style (round, butt, square).
na.rm: If `FALSE`, the default, missing values are removed with a warning. If `TRUE`, missing values are silently removed.
show.legend: logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes. It can also be a named logical vector to finely select the aesthetics to display.
inherit.aes: If `FALSE`, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn’t inherit behaviour from the default plot specification, e.g. `borders()`.
geom: The geometric object to use display the data

Details

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

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

The starting grid is a semi regular grid defined, either by the resolution of the field and the `skip.x` and `skip.y` parameters 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
Geom Streamline

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

```r
## 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]
```
## geopotential

# 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.

```r
ggplot(topo, aes(lon, lat)) +
  geom_relief(aes(z = h), interpolate = TRUE, data = topo[h >= 0]) +
  geom_contour(aes(z = h), breaks = 0, color = "black") +
  geom_streamline(aes(dx = -dx, dy = -dy), L = 10, skip = 3, arrow = NULL,
                  color = "#4658BD") +
  coord_quickmap()

## End(Not run)
```

### geopotential

**Geopotential height**

---

**Description**

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

**Usage**

geopotential

**Format**

A data.table with 53224 rows and 5 variables.

- **lon**: longitude in degrees
- **lat**: latitude in degrees
- **lev**: level in hPa
- **gh**: geopotential height in meters
- **date**: date

**Source**

[https://www.esrl.noaa.gov/psd/data/gridded/data.ncep.reanalysis.derived.pressure.html](https://www.esrl.noaa.gov/psd/data/gridded/data.ncep.reanalysis.derived.pressure.html)
**GeostrophicWind**

*Calculate geostrophic winds*

**Description**

Geostrophic wind from a geopotential height field.

**Usage**

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

**Arguments**

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

**Details**

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

**Value**

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

**See Also**

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

**Examples**

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

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

Usage

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

Arguments

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

Value

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

    date  date
    t  temperature in degrees celsius
    rh  relative humidity in %
    slp  sea level pressure in hPa
    dir  wind direction in clockwise degrees from 6 o’clock
    V  wind magnitude in m/s
    station  station name

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

    date  date
    tmax  maximum daily temperature in degrees celsius
    tmin  minimum daily temperature in degrees celsius
**GetTopography**

*Get topographic data*

**Description**

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

**Usage**

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

---

station  station name

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

date  date

global  global radiation in W/m^2
diffuse  diffuse radiation in W/m^2
station  station name

**Source**

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

**Examples**

```r
## 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)
```
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: Proce-
dures, 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, verbose = FALSE)
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

```r
guide_vector(
  title = waiver(),
  title.position = NULL,
  title.theme = NULL,
  title.hjust = NULL,
  title.vjust = NULL,
  label = TRUE,
  label.position = NULL,
  label.theme = NULL,
  label.hjust = NULL,
  label.vjust = NULL,
  keywidth = NULL,
  keyheight = NULL,
  direction = NULL,
  default.unit = "cm",
  override.aes = list(),
  nrow = NULL,
  ncol = NULL,
  byrow = FALSE,
  reverse = FALSE,
  order = 0,
  ...
)
```

Arguments

title

A character string or expression indicating a title of guide. If NULL, the title is not shown. By default (waiver()), the name of the scale object or the name specified in labs() is used for the title.

title.position

A character string indicating the position of a title. One of "top" (default for a vertical guide), "bottom", "left" (default for a horizontal guide), or "right."

title.theme

A theme object for rendering the title text. Usually the object of element_text() is expected. By default, the theme is specified by legend.title in theme() or theme.

title.hjust

A number specifying horizontal justification of the title text.

title.vjust

A number specifying vertical justification of the title text.

label

logical. If TRUE then the labels are drawn. If FALSE then the labels are invisible.

label.position

A character string indicating the position of a label. One of "top", "bottom" (default for horizontal guide), "left", or "right" (default for vertical guide).

label.theme

A theme object for rendering the label text. Usually the object of element_text() is expected. By default, the theme is specified by legend.text in theme().

label.hjust

A numeric specifying horizontal justification of the label text.

label.vjust

A numeric specifying vertical justification of the label text.

keywidth

A numeric or a grid::unit() object specifying the width of the legend key. Default value is legend.key.width or legend.key.size in theme().
### Key Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>keyheight</code></td>
<td>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>.</td>
</tr>
<tr>
<td><code>direction</code></td>
<td>A character string indicating the direction of the guide. One of &quot;horizontal&quot; or &quot;vertical.&quot;</td>
</tr>
<tr>
<td><code>default.unit</code></td>
<td>A character string indicating <code>grid::unit()</code> for keywidth and keyheight.</td>
</tr>
<tr>
<td><code>override.aes</code></td>
<td>A list specifying aesthetic parameters of legend key. See details and examples.</td>
</tr>
<tr>
<td><code>nrow</code></td>
<td>The desired number of rows of legends.</td>
</tr>
<tr>
<td><code>ncol</code></td>
<td>The desired number of column of legends.</td>
</tr>
<tr>
<td><code>byrow</code></td>
<td>logical. If FALSE (the default) the legend-matrix is filled by columns, otherwise the legend-matrix is filled by rows.</td>
</tr>
<tr>
<td><code>reverse</code></td>
<td>logical. If TRUE the order of legends is reversed.</td>
</tr>
<tr>
<td><code>order</code></td>
<td>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.</td>
</tr>
<tr>
<td><code>...</code></td>
<td>ignored.</td>
</tr>
</tbody>
</table>

### See Also

- `scale_vector`

---

### Impute2D

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

### Description

Provides methods for (soft) imputation of missing values.

### Usage

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

### Arguments

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

### Details

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

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

*Impute missing values*

---

**Description**

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

**Usage**

```r
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**


**Examples**

```r
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**

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

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

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

# Interpolate multiple values
geopotential[, c("u", "v") := GeostrophicWind(gh, lon, lat)]
interpolated <- geopotential[, Interpolate(u | v ~ lon + lat, x.out, y.out)]
```
# Interpolate values following a path
lats <- c(-34, -54, -30)  # start and end latitudes
lons <- c(302, 290, 180)  # start and end longituded
path <- geopotential[, Interpolate(gh ~ lon + lat, as.path(lons, lats))]

---

**is.cross**

**Cross pattern**

Description

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

Usage

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

cross(x, y)

Arguments

x, y  

x and y points that define a regular grid.

skip  

how many points to skip. Greater value reduces the final point density.

Value

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

Examples

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

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

# Its intended use is to highlight areas with geom_subset()
# with reduced density. This "hatches" areas with temperature
# over 270K
library(ggplot2)
ggplot(temperature[lev == 500], aes(lon, lat)) +
  geom_raster(aes(fill = air)) +
  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
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 $\sqrt{x^2 + y^2 + ...}$.

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 recicling!
## Not run:
Mag(1, 1:2)

## End(Not run)
MakeBreaks  

Functions for making breaks

Description

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

Usage

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

AnchorBreaks(anchor = 0, binwidth = NULL, exclude = NULL)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>binwidth</td>
<td>width of breaks</td>
</tr>
<tr>
<td>bins</td>
<td>number of bins, used if binwidth = NULL</td>
</tr>
<tr>
<td>exclude</td>
<td>a vector of breaks to exclude</td>
</tr>
<tr>
<td>anchor</td>
<td>anchor value</td>
</tr>
</tbody>
</table>

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

```r
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)

<table>
<thead>
<tr>
<th>MaskLand</th>
<th>Mask</th>
</tr>
</thead>
</table>

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

```r
# 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)) +
g 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](https://github.com/eliocamp/metR)
- Report bugs at [https://github.com/eliocamp/metR/issues](https://github.com/eliocamp/metR/issues)
### Percentile

**Percentiles**

**Description**
Computes percentiles.

**Usage**

```r
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**

```r
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**

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

```r
GlanceNetCDF(file, ...)
```
Arguments

- **file**: source to read from. Must be one of:
  - A string representing a local file with read access.
  - A string representing a URL readable by `ncdf4::nc_open()`. (this includes DAP urls).
  - A netcdf object returned by `ncdf4::nc_open()`.
- **vars**: a character vector with the name of the variables to read. If NULL, then it read all the variables.
- **out**: character indicating the type of output desired
- **subset**: a list of subsetting objects. See below.
- **key**: if TRUE, returns a data.table keyed by the dimensions of the data.
- **...**: ignored. Is there for convenience so that a call to `ReadNetCDF()` can be also valid for `GlanceNetCDF()`.

Value

The return format is specified by `out`. It can be a data table in which each column is a variable and each row, an observation; an array with named dimensions; or a vector. Since it’s possible to return multiple arrays or vectors (one for each variable), for consistency the return type is always a list. Either of these two options are much faster than the first since the most time consuming part is the melting of the array returned by `ncdf4::ncvar_get`. `out = "vector"` is particularly useful for adding new variables to an existing data frame with the same dimensions.

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

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

Subsetting

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

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

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

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

```
subset = list(list(lat = -90:-70, lon = 0:60),
              list(lat = 70:90, lon = 300:360))
```
will return two contiguous chunks: one on the South-West corner and one on the North-East corner. Alternatively, if you want to get the four corners that are combination of those two conditions,

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

Both operations can be mixed together. So for example this

```r
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

```r
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**

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

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

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

## Not run:
# Using a DAP url
field <- ReadNetCDF(url, subset = list(M = 1,
```
reverselog_trans 57

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

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

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

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

Arguments

```r
...
```

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)
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

```r
scale_x_longitude(
  name = "",
  ticks = 30,
  breaks = seq(-180, 360, by = ticks),
  expand = c(0, 0),
  labels = LonLabel,
  trans = "identity",
  ...
)

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

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

circle_y_latitude(
  name = "",
  ticks = 30,
  breaks = seq(-90, 90, by = ticks),
  expand = c(0, 0),
  labels = LatLabel,
  ...
)
```
scale_x_level(name = "", expand = c(0, 0), trans = "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 waiver(), the default, the name of the scale is taken from the first mapping used for that aesthetic. If NULL, the legend title will be omitted.

ticks spacing between breaks

breaks One of:
• 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())

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

labels One of:
• NULL for no labels
• waiver() for the default labels computed by the transformation object
• A character vector giving labels (must be same length as breaks)
• A function that takes the breaks as input and returns labels as output

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

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

... Other arguments passed on to scale_(x|y)_continuous()

See Also

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

Examples

```r
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

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

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

```r
scale_dx_continuous(
  name = waiver(),
  labels = waiver(),
  max_size = 1,
  default_unit = "cm",
  max = waiver(),
  guide = guide_vector(),
  ...
)
```
Arguments

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

labels
   One of:
   - 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

max_size
   size of the arrow in centimetres

default_unit
   ignored

max
   magnitude of the reference arrow in data units. Will be the maximum value if waiver()

guide
   type of legend

... other arguments passed on to scale_(x|y)_continuous()

Examples

library(ggplot2)
g <- ggplot(seals, aes(long, lat)) + geom_vector(aes(dx = delta_long, dy = delta_lat), skip = 2)
g + scale_mag("Seals velocity")
g + scale_mag("Seals velocity", max = 1)
g + scale_mag("Seals velocity", max_size = 2)
g + scale_mag("Seals velocity", default_unit = "mm")

season
Assign seasons to months

Description
Assign seasons to months

Usage

season(x, lang = c("en", "es"))
seasonally(x)
is.full_season(x)
Arguments

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

Value

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

Examples

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

```
dlons(dx, lat, a = 6731000)
dlats(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

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

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

stat_na

Filter only NA values.

Description

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

Usage

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

Arguments

- **mapping**: Set of aesthetic mappings created by `aes()` or `aes()`. If specified and `inherit.aes` = `TRUE` (the default), it is combined with the default mapping at the top level of the plot. You must supply `mapping` if there is no plot mapping.
- **data**: The data to be displayed in this layer. There are three options: If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`. A `data.frame`, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify()` for which variables will be created. 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. \(~ \text{head}(.x, 10))
- **geom**: The geometric object to use display the data
Position adjustment, either as a string, or the result of a call to a position adjustment function.

... Other arguments passed on to layer(). These are often aesthetics, used to set an aesthetic to a fixed value, like colour = "red" or size = 3. They may also be parameters to the paired geom/stat.

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.

If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn’t inherit behaviour from the default plot specification, e.g. borders().

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")
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 \texttt{aes()} or \texttt{aes().} If specified and \texttt{inherit.aes =} TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply \texttt{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 \texttt{ggplot().}
A \texttt{data.frame}, or other object, will override the plot data. All objects will be fortified to produce a data frame. See \texttt{fortify()} for which variables will be created.
A function will be called with a single argument, the plot data. The return value must be a \texttt{data.frame}, and will be used as the layer data. A function can be created from a formula (e.g. \texttt{~ head(.x,10)}).

geom The geometric object to use display the data

position Position adjustment, either as a string, or the result of a call to a position adjustment function.

... Other arguments passed on to \texttt{layer().} These are often aesthetics, used to set an aesthetic to a fixed value, like \texttt{colour = "red"} or \texttt{size = 3}. They may also be parameters to the paired geom/stat.

show.legend logical. Should this layer be included in the legends? \texttt{NA}, the default, includes if any aesthetics are mapped. \texttt{FALSE} never includes, and \texttt{TRUE} always includes. It can also be a named logical vector to finely select the aesthetics to display.

inherit.aes If \texttt{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. \texttt{borders().}
Aesthetics

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

- x
- y
- subset
- width
- height

See Also

stat_na for a more specialized stat for filtering NA values.

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

Examples

```r
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")
```

Description

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

Usage

```r
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://www.esrl.noaa.gov/psd/data/gridded/data.ncep.reanalysis.derived.pressure.html

thermodynamics  Thermodynamics

Description

Functions related to common atmospheric thermodynamic relationships.

Usage

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

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

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

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

ClausiusClapeyron(t, es)

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

Arguments

p  pressure

h  temperature

t  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

tv  vapour partial pressure

e  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 $T(1 - e/P(1 - e))^{-1}$.

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

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 ratio or dew point from the relationship $ws = ees(Td)/(p - es(Td))$. Note that the computation of dew point is approximated.

It 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/courcelinks/fall11/atmo551a/ATMO_451a_551a_files/WaterVapor.pdf

See Also

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

Examples

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

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

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

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

Compute trajectories

Description

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

Usage

\texttt{Trajectory(formula, x0, y0, cyclical = FALSE, data = NULL, res = 2)}

Arguments

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

WaveFlux

Calculate wave-activity flux

Description

Calculate wave-activity flux

Usage

\texttt{WaveFlux(gh, u, v, lon, lat, lev, g = 9.81, a = 6371000)}

Arguments

- \texttt{gh}: geopotential height
- \texttt{u}: mean zonal velocity
- \texttt{v}: mean meridional velocity
- \texttt{lon}: longitude (in degrees)
- \texttt{lat}: latitude (in degrees)
- \texttt{lev}: pressure level (in hPa)
- \texttt{g}: acceleration of gravity
- \texttt{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


Adapted from https://github.com/marisolosman/Reunion_Clima/blob/master/WAF/Calculo_WAF.ipynb

See Also

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

---

waves  

Fourier transform

---

Description

Perform a fourier transform of the data and return the

Usage

FitWave(y, k = 1)

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

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

WaveEnvelope(y)
Arguments

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

Details

`FitWave` uses fft to make a fourier transform of the data and then returns a list of parameters for each wave number kept. The amplitude (A), phase (φ) 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 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


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.only12 := FilterWave(gh, -2:0), by = .(lat)]
ggplot(jan, aes(lon, lat)) +
  geom_contour(aes(z = gh.only12, 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 repeated 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

+Topic datasets
  geom_arrow, 14
  geom_contour2, 18
  geom_contour_fill, 21
  geom_contour_tanaka, 23
  geom_label_contour, 26
  geom_relief, 29
  geom_streamline, 31
  geopotential, 36
  stat_na, 66
  stat_subset, 68
  temperature, 69
  %-% (logic), 48
  %in%, 48

Adiabat (thermodynamics), 70
aes(), 16, 19, 22, 24, 27, 30, 33, 66, 68
aes_(), 16, 19, 22, 24, 27, 30, 33, 66, 68
AnchorBreaks (MakeBreaks), 50
Anomaly, 3, 47–49, 54
as.path, 4, 45
AssignSeason (season), 64
autoplot, 11

base::svd, 11
borders(), 16, 20, 22, 25, 28, 30, 34, 67, 68
BuildWave (waves), 73

ClausiusClapeyron (thermodynamics), 70
continuous_scale, 58
ConvertLongitude, 4
coord_cartesian(), 59
coriolis, 5
cross (is.cross), 46
cut.eof, 6, 11

data.table, 53
data.table::dcast, 43
denormalise, 6
denormalize (denormalise), 6

Derivate, 7, 11, 37, 53, 71, 73, 75
DewPoint (thermodynamics), 70
Divergence (Derivate), 7
DivideTimeseries, 9, 17, 20, 23, 28, 31, 35, 50, 51, 57, 60, 62, 67, 69, 76
dlat (spherical), 65
dlon (spherical), 65
dx (spherical), 65
dy (spherical), 65

element_text(), 41
EOF, 8, 10, 37, 53, 71, 73, 75
EOF(), 6
EPflux, 12
expansion(), 60, 62

f (coriolis), 5
fft, 74
fields::interp.surface, 44
FilterWave (waves), 73
FitLM, 13
FitWave (waves), 73
Formula::Formula, 11, 43
fortify(), 16, 19, 22, 24, 27, 30, 33, 66, 68

geom_arrow, 10, 14, 20, 23, 28, 31, 35, 50, 51, 57, 60, 62, 63, 67, 69, 76
geom_contour, 21
geom_contour2, 10, 17, 18, 23, 28, 31, 35, 50, 51, 57, 60, 62, 67, 69, 76
geom_contour_fill, 10, 17, 20, 21, 28, 31, 35, 50, 51, 57, 60, 62, 67, 69, 76
geom_contour_fill(), 42
geom_contour_tanaka, 23
geom_label_contour, 10, 17, 20, 23, 26, 31, 35, 50, 51, 57, 60, 62, 67, 69, 76
geom_relief, 10, 17, 20, 23, 28, 29, 35, 50, 51, 57, 60, 62, 67, 69, 76
geom_shadow (geom_relief), 29
geom_streamline, 10, 17, 20, 23, 28, 31, 50, 51, 57, 60, 62, 67, 69, 76
geom_text_contour (geom_label_contour), 26
geom_vector (geom_arrow), 14
GeomArrow (geom_arrow), 14
GeomContour2 (geom_contour2), 18
GeomContourTanaka (geom_contour_tanaka), 23
GeomLabelContour (geom_label_contour), 26
GeomRelief (geom_relief), 29
GeomShadow (geom_relief), 29
GeomStreamline (geom_streamline), 31
GeomTextContour (geom_label_contour), 26
geopotential, 36
GeostrophicWind, 8, 11, 37, 71, 73, 75
GetSMNData, 38
GetTopography, 39
ggplot(), 16, 19, 22, 24, 27, 30, 33, 36, 68
ggplot2, 53
ggplot2::geom_contour, 18, 21
ggplot2::geom_raster, 30
ggplot2::geom_segment, 14
ggplot2::geom_tile, 30
ggplot2::stat_contour, 26, 28, 30
GlanceNetCDF (ReadNetCDF), 54
GlanceNetCDF(), 55
ggrid::arrow, 16, 33, 34
ggrid::unit(), 41, 42
guide_colourstrip, 10, 17, 20, 23, 28, 31, 35, 50, 51, 57, 60, 62, 67, 69, 76
guide_vector, 40
IdealGas (thermodynamics), 70
Impute2D, 20, 22, 42
ImputeEOF, 43
Interpolate, 4, 44
irlba::irlba, 11
is.cross, 46
is.full_season (season), 64
JumpBy, 3, 47, 48, 49, 54
labs(), 41
Laplacian (Derivate), 7
LatLabel (map_labels), 51
layer(), 16, 19, 22, 24, 27, 30, 33, 67, 68
logic, 3, 47, 48, 49, 54
LonLabel (map_labels), 51
Mag, 3, 47, 48, 49, 54
MakeBreaks, 10, 17, 20, 23, 28, 31, 35, 50, 51, 57, 60, 62, 67, 69, 76
map, 52
map_labels, 10, 17, 20, 23, 28, 31, 35, 50, 51, 57, 60, 62, 67, 69, 76
MaskLand, 52
mean, 3, 42
metR, 53
metR-package (metR), 53
MixingRatio (thermodynamics), 70
ncdf4::nc_open(), 55
ncdf4::ncvar_get, 55
ncvar_get, 54
Percentile, 3, 47–49, 54
predict, 11
ReadNetCDF, 54
ReadNetCDF(), 55
RepeatCircular (WrapCircular), 76
reverselog_trans, 10, 17, 20, 23, 28, 31, 35, 50, 51, 57, 60, 62, 67, 69, 76
scale_color_divergent (scale_divergent), 58
scale_colour_divergent (scale_divergent), 58
scale_colour_gradient2, 58
scale_colour_gradient2(), 59
scale_colour_gradientn(), 59
scale_divergent, 10, 17, 20, 23, 28, 31, 35, 50, 51, 57, 58, 62, 67, 69, 76
scale_dx_continuous (scale_mag), 63
scale_fill_divergent (scale_divergent), 58
scale_latitude (scale_longitude), 61
scale_longitude, 10, 17, 20, 23, 28, 31, 35, 50, 51, 57, 60, 61, 67, 69, 76
scale_mag, 63
scale_mag_continuous (scale_mag), 63
scale_x_continuous, 61
scale_x_latitude (scale_longitude), 61
scale_x_level (scale_longitude), 61
scale_x_longitude (scale_longitude), 61
scale_y_continuous, 61
scale_y_latitude(scale_longitude), 61
scale_y_level, 53
scale_y_level(scale_longitude), 61
scale_y_longitude(scale_longitude), 61
scales::area_pal(), 59
scales::boxcox_trans(), 60, 62
scales::censor(), 59
scales::extended_breaks(), 59, 62
scales::rescale(), 59
scales::squish(), 59
scales::squish_infinite(), 59
scales::trans_new(), 60, 62
screeplot, 11
season, 64
seasonally (season), 64
Similar (logic), 48
spherical, 34, 65
stat_contour2(geom_contour2), 18
stat_contour_fill, 53
stat_contour_fill(geom_contour_fill), 21
stat_na, 10, 17, 20, 23, 28, 31, 35, 50, 51, 57, 60, 62, 66, 69, 76
stat_streamline(geom_streamline), 31
stat_subset, 10, 17, 20, 23, 28, 31, 35, 50, 51, 57, 60, 62, 67, 68, 76
StatArrow(geom_arrow), 14
StatContour2(geom_contour2), 18
StatContourFill(geom_contour_fill), 21
StatNa(stat_na), 66
stats::.lm.fit, 13
StatStreamline(geom_streamline), 31
StatSubset(stat_subset), 68
StatTextContour(geom_label_contour), 26
summary, 11

temperature, 69
theme(), 41, 42
thermodynamics, 8, 11, 37, 70, 73, 75
Trajectory, 72
transformation object, 59, 62

varimax, 11
VirtualTemperature(thermodynamics), 70
Vorticity(Derivate), 7

waiver(), 41
WaveEnvelope(waves), 73
WaveFlux, 8, 11, 37, 71, 72, 75