Package 'RmarineHeatWaves'

January 20, 2025

Version 0.17.0

Date 2018-06-04

Title Detect Marine Heat Waves and Marine Cold Spells

Type Package

Maintainer Albertus J. Smit <albertus.smit@gmail.com>

Depends R (>= 3.00)

Imports tibble, ggplot2, lubridate, dplyr, stats, utils, zoo, tidyr, plyr, raster, grid, lazyeval, rlang

Suggests knitr, rmarkdown

VignetteBuilder knitr

Description Given a time series of daily temperatures, the package provides tools to detect extreme thermal events, including marine heat waves, and to calculate the exceedances above or below specified threshold values. It outputs the properties of all detected events and exceedances.

License MIT + file LICENSE

URL https://github.com/ajsmit/RmarineHeatWaves

LazyData TRUE

RoxygenNote 6.0.1

NeedsCompilation no

Author Albertus J. Smit [aut, cre] (R implementation.), Eric C. J. Oliver [aut] (The brain behind the Python implementation.), Robert W. Schlegel [ctb] (Graphical and data summaries.)

Repository CRAN

Date/Publication 2018-06-04 17:43:40 UTC

Contents

`block_average` *Calculate Yearly Means for Event Metrics.*

Description

Calculate Yearly Means for Event Metrics.

Usage

```
block_average(data, x = t, y = temp, report = "full")
```

Arguments

`data`	Accepts the data returned by the `detect` function.
`x`	This column is expected to contain a vector of dates as per the specification of `make_whole`. If a column headed `t` is present in the dataframe, this argument may be ommitted; otherwise, specify the name of the column with dates here.
`y`	This is a column containing the measurement variable. If the column name differs from the default (i.e. `temp`), specify the name here.
`report`	Specify either `full` or `partial`. Selecting `full` causes the report to contain NAs for any years in which no events were detected (except for `count`, which will be zero in those years), while `partial` reports only the years wherein events were detected. The default is `full`.

Details

This function needs to be provided with the full output from the `detect` function. Note that the yearly averages are calculted only for complete years (i.e. years that start/end part-way through the year at the beginning or end of the original time series are removed from the calculations).

This function differs from the python implementation of the function of the same name (i.e., `blockAverage`, see https://github.com/ecjoliver/marineHeatWaves) in that we only provide the ability to calculate the average (or aggregate) event metrics in 'blocks' of one year, while the python version allows arbitrary (integer) block sizes.

Value

The function will return a data frame of the averaged (or aggregate) metrics. It includes the following:

`year`	The year over which the metrics were averaged.
`temp_mean`	Seawater temperature for the specified year [deg. C].
`temp_min`	The minimum temperature for the specified year [deg. C].
`temp_max`	The maximum temperature for the specified year [deg. C].
`count`	The number of events per year.
`duration`	The average duration of events per year [days].
`int_mean`	The average event "mean intensity" in each year [deg. C].
`int_max`	The average event "maximum (peak) intensity" in each year [deg. C].
`int_var`	The average event "intensity variability" in each year [deg. C].
`int_cum`	The average event "cumulative intensity" in each year [deg. C x days].
`rate_onset`	Average event onset rate in each year [deg. C / days].
`rate_decline`	Average event decline rate in each year [deg. C / days].
`total_days`	Total number of events days in each year [days].
`total_icum`	Total cumulative intensity over all events in each year [deg. C x days].

`int_max_rel_thresh`, `int_mean_rel_thresh`, `int_var_rel_thresh`, and `int_cum_rel_thresh` are as above except relative to the threshold (e.g., 90th percentile) rather than the seasonal climatology.

`int_max_abs`, `int_mean_abs`, `int_var_abs`, and `int_cum_abs` are as above except as absolute magnitudes rather than relative to the seasonal climatology or threshold.

`int_max_norm` and `int_mean_norm` are as above except units are in multiples of threshold exceedances, i.e., a value of 1.5 indicates the event intensity (relative to the climatology) was 1.5 times the value of the threshold (relative to climatology, i.e., threshold - climatology.)

Author(s)

Albertus J. Smit, Eric C. J. Oliver

References

Hobday, A.J. et al. (2016), A hierarchical approach to defining marine heatwaves, Progress in Oceanography, 141, pp. 227-238, doi: 10.1016/j.pocean.2015.12.014

Examples

```
# ts_dat <- make_whole(sst_Med)
# res <- detect(ts_dat, climatology_start = "1983-01-01",
#               climatology_end = "2012-12-31")
# out <- block_average(res)
# summary(glm(count ~ year, out, family = "poisson"))
```

```
## Not run:
plot(out$year, out$count, col = "salmon", pch = 16,
     xlab = "Year", ylab = "Number of events")
lines(out$year, out$count)

## End(Not run)
```

`detect` *Detect heatwaves and cold-spells.*

Description

Applies the Hobday et al. (2016) marine heat wave definition to an input time series of temperature along with a daily date vector.

Usage

```
detect(data, doy = doy, x = t, y = temp, climatology_start,
  climatology_end, pctile = 90, window_half_width = 5,
  smooth_percentile = TRUE, smooth_percentile_width = 31,
  clim_only = FALSE, min_duration = 5, join_across_gaps = TRUE,
  max_gap = 2, max_pad_length = 3, cold_spells = FALSE)
```

Arguments

`data`	A data frame with three columns. In the default setting (i.e. ommitting the arguments `doy`, `x` and `y`; see immediately below), the data set is expected to have the headers `doy`, `t` and `temp`. `doy` is the Julian day running from 1 to 366, but modified so that the day-of-year (doy) vector for non-leap-years runs 1...59 and then 61...366. For leap years the 60th day is February 29. The `t` column is a vector of dates of class `Date`, while `temp` is the measured variable (by default it is assumed to be temperature). Data of the appropriate format are created by the function `make_whole`, but your own data can be supplied if they meet the criteria specified by `make_whole`.
`doy`	If a column headed `doy` is not available, another column with Julian dates can be supplied. This argument accepts the name of that column. The default name is, of course, `doy`.
`x`	This column is expected to contain a vector of dates as per the specification of `make_whole`. If a column headed `t` is present in the dataframe, this argument may be ommitted; otherwise, specify the name of the column with dates here.
`y`	This is a column containing the measurement variable. If the column name differs from the default (i.e. `temp`), specify the name here.
`climatology_start`	Required. The start date for the period across which the (varying by day-of-year) seasonal cycle and extremes threshold are calculated.

`climatology_end`
: Required. The end date for the period across which the (varying by day-of-year) seasonal cycle and extremes threshold are calculated.

`pctile`
: Threshold percentile (%) for detection of extreme values. Default is `90`th percentile. Please see `cold_spells` for more information about the calculation of marine cold spells.

`window_half_width`
: Width of sliding window about day-of-year (to one side of the center day-of-year) used for the pooling of values and calculation of climatology and threshold percentile. Default is `5` days, which gives a window width of 11 days centered on the 6th day of the series of 11 days.

`smooth_percentile`
: Boolean switch selecting whether to smooth the climatology and threshold percentile timeseries with a moving average of width `smooth_percentile`. Default is `TRUE`.

`smooth_percentile_width`
: Full width of moving average window for smoothing climatology and threshold. Default is `31` days.

`clim_only`
: Choose to calculate only the climatologies and not the events. Default is `FALSE`.

`min_duration`
: Minimum duration for acceptance of detected MHWs. Default is `5` days.

`join_across_gaps`
: Boolean switch indicating whether to join MHWs which occur before/after a short gap as specified by `max_gap`. Default is `TRUE`.

`max_gap`
: Maximum length of gap allowed for the joining of MHWs. Default is `2` days.

`max_pad_length`
: Specifies the maximum length of days over which to interpolate (pad) missing data (specified as `NA`) in the input temperature time series; i.e., any consecutive blocks of NAs with length greater than `max_pad_length` will be left as `NA`. Set as an integer. Default is `3` days.

`cold_spells`
: Boolean specifying if the code should detect cold events instead of heat events. Default is `FALSE`. Please note that the climatological thresholds for cold-spells are calculated the same as for heatwaves, meaning that `pctile` should be set the same regardless if one is calculating heatwaves or cold-spells. For example, if one wants to calculate heatwaves above the 90th percentile threshold (the default) one sets `pctile = 90`. Likewise, if one would like identify the most intense cold-spells one must also set `pctile = 90`, even though cold spells are in fact simply the coldest extreme events in a time series, which statistically equate to values below the 10th percentile.

Details

1. This function assumes that the input time series consists of continuous daily values with few missing values. Time ranges which start and end part-way through the calendar year are supported. The accompanying function `make_whole` aids in the preparation of a time series that is suitable for use with `detect`, although this may also be accomplished 'by hand' as long as the criteria are met as discussed in the documentation to `make_whole`.

2. It is recommended that a climatology period of at least 30 years is specified in order to capture decadal thermal periodicities. It is further advised that full the start and end dates for the climatology period result in full years, e.g. "1982-01-01" to "2011-12-31" or "1982-07-01" to "2012-06-30"; if not, this may result in an unequal weighting of data belonging with certain months within a time series.
3. This function supports leap years. This is done by ignoring Feb 29s for the initial calculation of the climatology and threshold. The values for Feb 29 are then linearly interpolated from the values for Feb 28 and Mar 1.
4. The calculation of onset and decline rates assumes that the events started a half-day before the start day and ended a half-day after the end-day. This is consistent with the duration definition as implemented, which assumes duration = end day - start day + 1. As of version 0.15.7, an event that is already present at the beginning of a time series, or an event that is still present at the end of a time series, will report the rate of onset or the rate of decline as `NA`, as it is impossible to know what the temperature half a day before or after the start or end of the event is. This may be a departure from the python marineHeatWaves function.
5. For the purposes of event detection, any missing temperature values not interpolated over (through optional `max_pad_length`) will be set equal to the seasonal climatology. This means they will trigger the end/start of any adjacent temperature values which satisfy the event definition criteria.
6. If the code is used to detect cold events (`coldSpells` = TRUE), then it works just as for heat waves except that events are detected as deviations below the (100 - pctile)th percentile (e.g., the 10th instead of 90th) for at least 5 days. Intensities are reported as negative values and represent the temperature anomaly below climatology.
7. If only the climatology for the time series is required, and not the events themselves, this may be done by setting `clim_only` = TRUE.

The original Python algorithm was written by Eric Oliver, Institute for Marine and Antarctic Studies, University of Tasmania, Feb 2015, and is documented by Hobday et al. (2016). The marine cold spell option was implemented in version 0.13 (21 Nov 2015) of the Python module as a result of our preparation of Schlegel et al. (submitted), wherein the cold events receive a brief overview.

Value

The function will return a list of two tibbles (see the `tidyverse`), `clim` and `event`, which are the climatology and events, respectively. The climatology contains the full time series of daily temperatures, as well as the the seasonal climatology, the threshold and various aspects of the events that were detected. The software was designed for detecting extreme thermal events, and the units specified below reflect that intended purpose. However, the various other kinds of extreme events may be detected according to the 'marine heat wave' specifications, and if that is the case, the appropriate units need to be determined by the user.

`doy` Julian day (day-of-year). For non-leap years it runs 1...59 and 61...366, while leap years run 1...366. This column will be named differently if another name was specified to the `doy` argument.

`t` The date of the temperature measurement. This column will be named differently if another name was specified to the `x` argument.

`temp` If the software was used for the purpose for which it was designed, seawater temperature [deg. C] on the specified date will be returned. This column will of

course be named differently if another kind of measurement was specified to the y argument.

`seas_clim_year` Climatological seasonal cycle [deg. C].

`thresh_clim_year`
Seasonally varying threshold (e.g., 90th percentile) [deg. C].

`var_clim_year` Seasonally varying variance (standard deviation) [deg. C].

`thresh_criterion`
Boolean indicating if `temp` exceeds `thresh_clim_year`.

`duration_criterion`
Boolean indicating whether periods of consecutive `thresh_criterion` are >= `min_duration`.

`event` Boolean indicating if all criteria that define a MHW or MCS are met.

`event_no` A sequential number indicating the ID and order of occurence of the MHWs or MCSs.

The events are summarised using a range of event metrics:

`index_start` Start index of event.

`index_stop` Stop index of event.

`event_no` A sequential number indicating the ID and order of the events.

`duration` Duration of event [days].

`date_start` Start date of event [date].

`date_stop` Stop date of event [date].

`date_peak` Date of event peak [date].

`int_mean` Mean intensity [deg. C].

`int_max` Maximum (peak) intensity [deg. C].

`int_var` Intensity variability (standard deviation) [deg. C].

`int_cum` Cumulative intensity [deg. C x days].

`rate_onset` Onset rate of event [deg. C / day].

`rate_decline` Decline rate of event [deg. C / day].

`int_max_rel_thresh`, `int_mean_rel_thresh`, `int_var_rel_thresh`, and `int_cum_rel_thresh` are as above except relative to the threshold (e.g., 90th percentile) rather than the seasonal climatology.

`int_max_abs`, `int_mean_abs`, `int_var_abs`, and `int_cum_abs` are as above except as absolute magnitudes rather than relative to the seasonal climatology or threshold.

`int_max_norm` and `int_mean_norm` are as above except units are in multiples of threshold exceedances, i.e., a value of 1.5 indicates the event intensity (relative to the climatology) was 1.5 times the value of the threshold (relative to climatology, i.e., threshold - climatology.)

Note that `rate_onset` and `rate_decline` will return `NA` when the event begins/ends on the first/last day of the time series. This may be particularly evident when the function is applied to large gridded data sets. Although the other metrics do not contain any errors and provide sensible values, please take this into account in its interpretation.

Author(s)

Albertus J. Smit, Robert W. Schlegel, Eric C. J. Oliver

References

Hobday, A.J. et al. (2016). A hierarchical approach to defining marine heatwaves, Progress in Oceanography, 141, pp. 227-238, doi:10.1016/j.pocean.2015.12.014

Schlegel, R. W., Oliver, C. J., Wernberg, T. W., Smit, A. J. (2017). Coastal and offshore co-occurrences of marine heatwaves and cold-spells. Progress in Oceanography, 151, pp. 189-205, doi:10.1016/j.pocean.2017.01.004

Examples

```
ts_dat <- make_whole(sst_WA)
res <- detect(ts_dat, climatology_start = "1983-01-01",
              climatology_end = "2012-12-31")
# show a portion of the climatology:
res$clim[1:10, ]
# show some of the heat waves:
res$event[1:5, 1:10]
```

---

event_line *Create a Line Plot of Marine Heat Waves or Cold Spells.*

Description

Creates a graph of warm or cold events as per the second row of Figure 3 in Hobday et al. (2016).

Usage

```
event_line(data, x = t, y = temp, min_duration = 5, spread = 150,
  metric = "int_cum", start_date, end_date)
```

Arguments

data	The function receives the output from the detect function.
x	This column is expected to contain a vector of dates as per the specification of make_whole. If a column headed t is present in the dataframe, this argument may be ommitted; otherwise, specify the name of the column with dates here.
y	This is a column containing the measurement variable. If the column name differs from the default (i.e. temp), specify the name here.
min_duration	The minimum duration that an event has to for it to qualify as a marine heat wave or marine cold spell.
spread	The the number of days leading and trailing the largest event (as per metric) detected within the time period specified by start_date and end_date. The default is 150 days.

`metric` One of the following options: `int_mean`, `int_max`, `int_var`, `int_cum`, `int_mean_rel_thresh`, `int_max_rel_thresh`, `int_var_rel_thresh`, `int_cum_rel_thresh`, `int_mean_abs`, `int_max_abs`, `int_var_abs`, `int_cum_abs`, `int_mean_norm`, `int_max_norm`, `rate_onset`, `rate_decline`. Partial name matching is currently not supported so please specify the metric name precisely. The default is `int_cum`.

`start_date` The start date of a period of time within which the largest event (as per `metric`) is retrieved and plotted. This may not necessarily correspond to the biggest event of the specified metric within the entire data set. To plot the biggest event within the whole time series, make sure `start_date` and `end_date` straddle this event, or simply specify the start and end dates of the full time series given to `detect`.

`end_date` The end date of a period of time within which the largest event (as per `metric`) is retrieved and plotted. See `start_date` for additional information.

Value

The function will return a line plot indicating the climatology, threshold and temperature, with the hot or cold events that meet the specifications of Hobday et al. (2016) shaded in as appropriate. The plotting of hot or cold events depends on which option is specified in `detect`. The top event detect during the selected time period will be visible in a brighter colour. This function differs in use from `geom_flame` in that it creates a stand alone figure. The benefit of this being that one must not have any prior knowledge of ggplot2 to create the figure.

Author(s)

Robert W. Schlegel

References

Hobday, A.J. et al. (2016), A hierarchical approach to defining marine heatwaves, Progress in Oceanography, 141, pp. 227-238, doi: 10.1016/j.pocean.2015.12.014

Examples

```
ts_dat <- make_whole(sst_WA)
res <- detect(ts_dat, climatology_start = "1983-01-01",
              climatology_end = "2012-12-31")

## Not run:
event_line(res, spread = 200, metric = "int_cum",
start_date = "2010-10-01", end_date = "2011-08-30")

## End(Not run)
```

exceedance *Detect consecutive days in exceedance of a given threshold.*

Description

Detect consecutive days in exceedance of a given threshold.

Usage

```
exceedance(data, x = t, y = temp, threshold = 20, below = FALSE,
  min_duration = 5, join_across_gaps = TRUE, max_gap = 2,
  max_pad_length = 3)
```

Arguments

data	A data frame with at least the two following columns: a `t` column which is a vector of dates of class `Date`, and a `temp` column, which is the temperature on those given dates. If columns are named differently, their names can be supplied as `x` and `y` (see below). The function will not accurately detect consecutive days of temperatures in exceedance of the `threshold` if missing days of data are not filled in with `NA`. Data of the appropriate format are created by the function `make_whole`, but your own data may be used directly if they meet the given criteria.
x	This column is expected to contain a vector of dates as per the specification of `make_whole`. If a column headed `t` is present in the dataframe, this argument may be ommitted; otherwise, specify the name of the column with dates here.
y	This is a column containing the measurement variable. If the column name differs from the default (i.e. `temp`), specify the name here.
threshold	The static threshold used to determine how many consecutive days are in exceedance of the temperature of interest. Default is `20` degrees.
below	Default is `FALSE`. When set to TRUE, consecutive days of temperature below the `threshold` variable are calculated. When set to FALSE, consecutive days above the `threshold` variable are calculated.
min_duration	Minimum duration that temperatures must be in exceedance of the `threshold` variable. Default is `5` days.
join_across_gaps	A TRUE/FALSE statement that indicates whether or not to join consecutive days of temperatures in exceedance of the `threshold` across a small gap between groups before/after a short gap as specified by `max_gap`. Default is `TRUE`.
max_gap	The maximum length of the gap across which to connect consecutive days in exceedance of the `threshold` when `join_across_gaps` is `TRUE`.
max_pad_length	Specifies the maximum length of days over which to interpolate (pad) missing data (specified as `NA`) in the input temperature time series; i.e., any consecutive blocks of NAs with length greater than `max_pad_length` will be left as `NA`. Set as an integer. Default is `3` days.

Details

1. This function assumes that the input time series consists of continuous daily temperatures, with few missing values. The accompanying function `make_whole` aids in the preparation of a time series that is suitable for use with `exceedance`, although this may also be accomplished 'by hand' as long as the criteria are met as discussed in the documentation to `make_whole`.
2. Future versions seek to accomodate monthly and annual time series, too.
3. The calculation of onset and decline rates assumes that exceedance of the `threshold` started a half-day before the start day and ended a half-day after the end-day. This is consistent with the duration definition as implemented, which assumes duration = end day - start day + 1.
4. For the purposes of exceedance detection, any missing temperature values not interpolated over (through optional `max_pad_length`) will remain as `NA`. This means they will trigger the end of an exceedance if the adjacent temperature values are in exceedance of the `threshold`.
5. If the function is used to detect consecutive days of temperature under the given `theshold`, these temperatures are then taken as being in exceedance below the `threshold` as there is no antonym in the English language for 'exceedance'.

This function is based largely on the `detect` function found in this package, which was ported from the Python algorithm that was written by Eric Oliver, Institute for Marine and Antarctic Studies, University of Tasmania, Feb 2015, and is documented by Hobday et al. (2016).

Value

The function will return a list of two components. The first being `threshold`, which shows the daily temperatures and on which specific days the given `threshold` was exceeded. The second component of the list is `exceedance`, which shows a medley of statistics for each discrete group of days in exceedance of the given `threshold`. Note that any additional columns left in the data frame given to this function will be output in the `threshold` component of the output. For example, if one uses `make_whole` to prepare a time series for analysis and leaves in the `doy` column, this column will appear in the output.

The information shown in the `threshold` component is:

`t`	The date of the temperature measurement. This variable may named differently if an alternative name is supplied to the function's `x` argument.
`temp`	Temperature on the specified date [deg. C]. This variable may named differently if an alternative name is supplied to the function's `y` argument.
`thresh`	The static `threshold` chosen by the user [deg. C].
`thresh_criterion`	Boolean indicating if `temp` exceeds `threshold`.
`duration_criterion`	Boolean indicating whether periods of consecutive `thresh_criterion` are >= `min_duration`.
`exceedance`	Boolean indicting if all criteria that define a discrete group in exceedance of the `threshold` are met.
`exceedance_no`	A sequential number indicating the ID and order of occurence of exceedances.

The individual exceedances are summarised using the following metrics:

index_start	Row number on which exceedance starts.
index_stop	Row number on which exceedance stops.
exceedance_no	The same sequential number indicating the ID and order of the exceedance as found in the threshold component of the output list.
duration	Duration of exceedance [days].
date_start	Start date of exceedance [date].
date_stop	Stop date of exceedance [date].
date_peak	Date of exceedance peak [date].
int_mean	Mean intensity [deg. C].
int_max	Maximum (peak) intensity [deg. C].
int_var	Intensity variability (standard deviation) [deg. C].
int_cum	Cumulative intensity [deg. C x days].
rate_onset	Onset rate of exceedance [deg. C / day].
rate_decline	Decline rate of exceedance [deg. C / day].

int_max_abs, int_mean_abs, int_var_abs, and int_cum_abs are as above except as absolute magnitudes rather than relative to the threshold.

Author(s)

Robert W. Schlegel, Albertus J. Smit

Examples

```
ts_dat <- make_whole(sst_WA)
res <- exceedance(ts_dat, threshold = 25)
# show first ten days of daily data:
res$threshold[1:10, ]
# show first five exceedances:
res$exceedance[1:5, ]
```

---

geom_flame *Create 'Flame' Ploygons.*

Description

This function will create polygons between two lines. If given a temperature and theshold time series, like that produced by detect, the output will meet the specifications of Hobday et al. (2016) shown as 'flame polygons.' If one wishes to plot polygons below a given threshold, and not above, switch the values being fed to the y and y2 aesthetics. This function differs in use from event_line in that it must be created as a ggplot 'geom' object. The benefit of this being that one may add additional information to the figure as geom layers to ggplot2 graphs as may be necessary.

Usage

```
geom_flame(mapping = NULL, data = NULL, stat = "identity",
  position = "identity", ..., 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.
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 color = "red" or size = 3. They may also be parameters to the paired geom/stat.
na.rm	If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values.
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_flame understands the following aesthetics (required aesthetics are in bold):

- x
- y
- y2
- colour
- fill
- size
- alpha
- linetype

Author(s)

Robert W. Schlegel

References

Hobday, A.J. et al. (2016), A hierarchical approach to defining marine heatwaves, Progress in Oceanography, 141, pp. 227-238, doi: 10.1016/j.pocean.2015.12.014

See Also

event_line for a non-ggplot2 based flame function.

Examples

```
ts_dat <- make_whole(sst_WA)
res <- detect(ts_dat, climatology_start = "1983-01-01",
              climatology_end = "2012-12-31")
mhw <- res$clim
mhw <- mhw[10580:10690,]

## Not run:
require(ggplot2)
ggplot(mhw, aes(x = t, y = temp)) +
  geom_flame(aes(y2 = thresh_clim_year)) +
  geom_text(aes(x = as.Date("2011-02-01"), y = 28,
  label = "That's not a heatwave.\nThis, is a heatwave.")) +
  xlab("Date") + ylab(expression(paste("Temperature [", degree, "C]")))

## End(Not run)
```

---

geom_lolli *Visualise a Timeline of Several Event Metrics as 'Lollipops'.*

Description

The function will return a graph of the intensity of the selected metric along the *y*-axis versus a time variable along the *x*-axis. The number of top events (n) from the chosen metric may be highlighted in a brighter colour with the aesthetic value colour.n. This function differs in use from lolli_plot in that it must be created as a ggplot2 'geom' object. The benefit of this being that one may add additional information layer by layer to the figure as geoms as necessary.

Usage

```
geom_lolli(mapping = NULL, data = NULL, ..., n = 1, 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.
...	other arguments passed on to layer. These are often aesthetics, used to set an aesthetic to a fixed value, like color = "red" or size = 3. They may also be parameters to the paired geom/stat.
n	The number of top events to highlight. Default is 1. This parameter has no effect if colour.n is set to NA outside of aes().
na.rm	If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values.
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_lolli understands the following aesthetics (required aesthetics are in bold):

- x
- y
- alpha
- color
- linetype
- size
- shape
- stroke
- fill
- colour.n: While this value may be used as an aesthetic, it also works as a parameter for this function. If one chooses not to highlight any events, use colour.n = NA outside of aes(). One may also provide a non-static value to colour.na but remember that one may not provide multiple continuous or discrete scales to a single ggplot2 object. Therefore, if one provides a continuous value to aes(colour), the values supplied to colour.n must be discrete. ggplot2 will attempt to do this automatically.

Author(s)

Robert W. Schlegel

See Also

lolli_plot for a non-geom based lolliplot function.

Examples

```
ts_dat <- make_whole(sst_NW_Atl)
# with defaults:
res <- detect(ts_dat, climatology_start = "1983-01-01",
              climatology_end = "2012-12-31")
mhw <- res$event

## Not run:
require(lubridate)
# Height of lollis represent event durations and their colours
# are mapped to the events' cumulative intensity:
ggplot(mhw, aes(x = mhw$date_peak, y = mhw$duration)) +
  geom_lolli(n = 0, shape = 20, aes(colour = mhw$int_cum), colour.n = NA) +
  scale_color_distiller(palette = "Spectral", name = "Cumulative \nintensity") +
  xlab("Date") + ylab("Event duration [days]")

# Height of lollis represent event durations and the top three (longest)
# lollis are highlighted in red:
ggplot(mhw, aes(x = mhw$date_peak, y = mhw$duration)) +
  geom_lolli(n = 3, shape = 20, colour.n = "red") +
  scale_color_distiller(palette = "Spectral", name = "Cumulative \nintensity") +
  xlab("Date") + ylab("Event duration [days]")

## End(Not run)
```

---

lolli_plot *Create a Timeline of Selected Event Metrics.*

Description

Visualise a timeline of several event metrics as 'lollipop' graphs.

Usage

```
lolli_plot(data, metric = "int_max", event_count = 3,
  xaxis = "date_start")
```

Arguments

data	Output from the detect function.
metric	One of int_mean, int_max, int_cum and duration. Default is int_cum.
event_count	The number of top events to highlight. Default is 3.
xaxis	One of event_no, date_start or date_peak. Default is date_start.

Value

The function will return a graph of the intensity of the selected metric along the y-axis versus either t or event_no. The number of top events as per event_count will be highlighted in a brighter colour. This function differs in use from geom_lolli in that it creates a stand alone figure. The benefit of this being that one must not have any prior knowledge of ggplot2 to create the figure.

Author(s)

Albertus J. Smit and Robert W. Schlegel

Examples

```
ts_dat <- make_whole(sst_NW_Atl)
res <- detect(ts_dat, climatology_start = "1983-01-01",
              climatology_end = "2012-12-31")

## Not run:
lolli_plot(res, metric = "int_cum", event_count = 3, xaxis = "date_peak")

## End(Not run)
```

---

make_whole *Constructs a Continuous, Uninterrupted Time Series of Temperatures.*

Description

Takes a series of dates and temperatures, and if irregular (but ordered), inserts missing dates and fills correpsonding temperatures with NAs.

Usage

```
make_whole(data, x = t, y = temp)
```

Arguments

data	A data frame with columns for date and temperature data. Ordered daily data are expected, and although missing values (NA) can be accommodated, the function is only recommended when NAs occur infrequently, preferably at no more than 3 consecutive days.

x A column with the daily time vector (see details). For backwards compatibility, the column is named `t` by default.

y A column with the response vector. RmarineHeatWaves version <= 0.15.9 assumed that this would be daily seawater temperatures, but as of version 0.16.0 it may be any arbitrary measurement taken at a daily frequency. The default remains temperature, and the default column name is therefore `temp`, again hopefully ensuring backwards compatibility.

Details

Upon import, the package uses 'zoo' and 'lubridate' to process the input date and temperature data. It reads in daily data with the time vector specified as either `POSIXct` or `Date` (e.g. "1982-01-01 02:00:00" or "1982-01-01"). The data may be an irregular time series, but date must be ordered. The function constructs a complete time series from the start date to the end date, and fills in the regions in the time series where temperature data are missing, with NAs in the temperature vector. There must only be one temperature value per day otherwise the function will fail. It is up to the user to calculate daily data from sub-daily measurements. Leap years are automatically accommodated by 'zoo'.

This function can handle some of missing days, but this is not a licence to actually use these data for the detection of anomalous thermal events. Hobday et al. (2016) recommend gaps of no more than 3 days, which may be adjusted by setting the `max_pad_length` argument of the `detect` function. The longer and more frequent the gaps become the lower the fidelity of the annual climatology and threshold that can be calculated, which will not only have repercussions for the accuracy at which the event metrics can be determined, but also for the number of events that can be detected.

It is recommended that a climatology period of at least 30 years is specified in order to capture any decadal thermal periodicities.

Value

The function will return a data frame with three columns. The column headed `doy` (day-of-year) is the Julian day running from 1 to 366, but modified so that the day-of-year series for non-leap-years runs 1...59 and then 61...366. For leap years the 60th day is February 29. See the example, below. The other two columns take the names of `x` and `y`, if supplied, or it will be `t` and `temp` in case the default values were used. The `x` (or `t`) column is a series of dates of class `Date`, while `y` (or `temp`) is the measured variable. This time series will be uninterrupted and continuous daily values between the first and last dates of the input data.

Author(s)

Smit, A. J.

Examples

```
require(dplyr); require(tidyr); require(lubridate)
ts_dat <- make_whole(sst_WA) # default columns "t" and "temp", in that order
clim_start <- "1983-01-01"
clim_end <- "2012-12-31"
ts_dat %>%
filter(t >= clim_start & t <= clim_end) %>%
```

```
mutate(t = year(t)) %>%
spread(t, temp) %>%
filter(doy >= 55 & doy <= 65)
```

RmarineHeatWaves *RmarineHeatWaves.*

Description

This package is an R implementation of the python script marineHeatWaves (https://github.com/ecjoliver/marineHeatWaves) written by Eric C. J. Oliver as part of the marine heat waves definition by Hobday et al. (2016).

Details

Although the title of the package refers to marine heat waves (MHW), it is equally capable of detecting marine cold spells (MCS). This functionality to detect cold events is also present in the python package, where it was implemented as a result of the publication Schlegel et al. (2017) that discusses the quantification and detection of anomalously cold events. As of release 0.16.0, the detect function may also be applied to time series of other natural phenomena which one might want to express in terms of the summary metrics outlined in the paper by Hobday et al. (2016).

The main function is the detection function detect which takes as input a time series of temperature (and a corresponding series of dates) and outputs a set of detected MHWs or MCS, as well as the climatological (varying by day-of-year) seasonal cycle and extremes threshold. There are various helper functions to fascilitate developing an uninterrupted time series of temperatures (e.g. make_whole) and some options to produce graphical summaries and representations of the detected events such as event_line and lolli_plot, or the ggplot2 equivalents, geom_flame and geom_lolli.

This package is demonstrated by applying the MHW definition to observed SST records and showing how it identifies three historical MHWs: the 2011 Western Australia event, the 2012 Northwest Atlantic event and the 2003 Mediterranean event. These data are included herewith.

One may also use the exceedance function to calculate consecutive days above or below a given static threshold. The output of this function is similar to detect.

Author(s)

Albertus J. Smit <<albertus.smit@gmail.com>>, Robert W. Schlegel, Eric C. J. Oliver

References

Hobday, A. J. et al. (2016), A hierarchical approach to defining marine heatwaves. Progress in Oceanography, 141, pp. 227-238, <DOI:10.1016/j.pocean.2015.12.014> (official citation for this package).

Schlegel, R. W., Oliver, E. C. J., Wernberg, T. W., Smit, A. J. (2017) Coastal and offshore co-occurrences of marine heatwaves and cold-spells. Progress in Oceanography, 151, pp. 189-205, <DOI:10.1016/j.pocean.2017.01.004>

sst_Med — *Optimally Interpolated 0.25 degree SST for the Mediterranean region.*

Description

A dataset containing the sea surface temperature (in degrees Celsius) and date for the Mediterranean region from 1982-01-01 to 2014-12-31.

Usage

```
sst_Med
```

Format

A data frame with 12053 rows and 2 variables:

t date, as POSIXct

temp SST, in degrees Celsius ...

Source

https://www.ncdc.noaa.gov/oisst

sst_NW_Atl — *Optimally Interpolated 0.25 degree SST for the NW Atlantic region.*

Description

A dataset containing the sea surface temperature (in degrees Celsius) and date for the Northwest Atlantic region from 1982-01-01 to 2014-12-31.

Usage

```
sst_NW_Atl
```

Format

A data frame with 12053 rows and 2 variables:

t date, as POSIXct

temp SST, in degrees Celsius ...

Source

https://www.ncdc.noaa.gov/oisst

`sst_WA` *Optimally Interpolated 0.25 degree SST for the Western Australian region.*

Description

A dataset containing the sea surface temperature temperature (in degrees Celsius) and date for the Western Australian region for the period 1982-01-01 to 2014-12-31.

Usage

```
sst_WA
```

Format

A data frame with 12053 rows and 2 variables:

t date, as POSIXct

temp SST, in degrees Celsius ...

Source

https://www.ncdc.noaa.gov/oisst

Index