Package 'SEI'

January 26, 2024

Title Calculating Standardised Indices

Version 0.1.1

Description Convert a time series of observations to a time series of standardised indices that can be used to monitor variables on a common and probabilistically interpretable scale. The indices can be aggregated and rescaled to different time scales, visualised using plot capabilities, and calculated using a range of distributions. This includes flexible non- and semi-parametric methods, as suggested by Allen and Otero (2023) <doi:10.1016/j.renene.2023.119206>.

URL `https://github.com/noeliaof/SEI`

License GPL (>= 2)

Encoding UTF-8

RoxygenNote 7.2.3

Depends R (>= 3.5.0)

Imports ggplot2, xts, zoo, fitdistrplus

Suggests knitr, lubridate, dplyr, gridExtra, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

LazyData true

NeedsCompilation no

Author Sam Allen [aut, cre],
Noelia Otero [aut]

Maintainer Sam Allen <`sam.allen@unibe.ch`>

Repository CRAN

Date/Publication 2024-01-26 16:20:02 UTC

R topics documented:

`aggregate_xts` *Aggregate values in xts objects*

Description

Inputs an xts time series and outputs an xts time series whose values have been aggregated over a moving window of a user-specified length.

Usage

```
aggregate_xts(
  x,
  len,
  scale = c("days", "hours", "weeks", "quarters", "years"),
  fun = "sum",
  timescale = c("days", "hours", "weeks", "quarters", "years"),
  na_thres = 10
)
```

Arguments

`x`	xts object to be aggregated.
`len`	length of the aggregation period.
`scale`	timescale of the aggregation period, default is 'days'.
`fun`	function to apply to the aggregated data, default is 'sum'.
`timescale`	timescale of `x`, default is 'days'.
`na_thres`	threshold for the percentage of NA values allowed in the aggregation period, default = 10.

Details

This has been adapted from code available at `https://github.com/WillemMaetens/standaRdized`.

`len` is a single numeric value specifying over how many time units the data `x` is to be aggregated. By default, `len` is assumed to correspond to a number of days, but this can also be specified manually using the argument `scale`. `scale` must be one of: "days", "weeks", "months", "quarters", and "years".

`fun` determines the function used to aggregate the time series. By default, `fun = "sum"`, meaning the aggregation results in accumulations over the aggregation period. Alternative functions can also be used. For example, specifying `fun = "mean"` would return the mean over the aggregation period.

`timescale` is the timescale of the input data `x`. By default, this is assumed to be "days".

Since the time series `x` aggregates data over the aggregation period, problems may arise when `x` contains missing values. For example, if interest is on daily accumulations, but 50% of the values in the aggregation period are missing, the accumulation over this aggregation period will not be accurate. This can be controlled using the argument `na_thres`. `na_thres` specifies the percentage of NA values in the aggregation period before a NA value is returned. i.e. the proportion of values that are allowed to be missing. The default is `na_thres = 10`.

Value

An xts time series with aggregated values.

Author(s)

Sam Allen, Noelia Otero

Examples

```
data(data_supply, package = "SEI")
# consider hourly German energy production data in 2019
supply_de <- subset(data_supply, country == "Germany", select = c("date", "PWS"))
supply_de <- xts::xts(supply_de$PWS, order.by = supply_de$date)

# daily accumulations
supply_de_daily <- aggregate_xts(supply_de, len = 1, timescale = "hours")

# weekly means
supply_de_weekly <- aggregate_xts(supply_de, len = 1, scale = "weeks", fun = "mean", "hours")

plot(supply_de, main = "Hourly energy production in Germany")
plot(supply_de_daily, main = "Daily energy production in Germany")
plot(supply_de_weekly, main = "Weekly energy production in Germany")
```

`data_supply` *Time series of wind and solar energy production*

Description

This dataset contains hourly time series of wind and solar energy production in 27 European countries in 2019.

Usage

```
data("data_supply")
```

Format

An object of type `data.frame` containing 3 variables:

date A `POSIXct` series of times at which energy production is available.

country The country to which the energy production measurement corresponds.

PWS The hourly wind and solar energy production for the corresponding time and country.

Details

The dataframe `data_supply` contains 236520 (24 x 365 x 27) rows, containing the wind and solar energy production for each hour in 2019 for each of the 27 countries.

This corresponds to a subset of the data used in Bloomfield and Brayshaw. (2021), which can be accessed at https://researchdata.reading.ac.uk/321/. Users are referred to this paper for further details.

References

Bloomfield, Hannah and Brayshaw, David (2021): ERA5 derived time series of European aggregated surface weather variables, wind power, and solar power capacity factors: hourly data from 1950-2020. doi:10.17864/1947.000321

Examples

```
data("data_supply")
```

`data_wind_de` *Time series of average wind speed in Germany*

Description

This dataset contains a daily time series of average wind speeds across Germany between 1979 and 2019.

Usage

```
data("data_wind_de")
```

Format

An object of type `data.frame` containing 2 variables:

date A `POSIXct` series of times at which average wind speeds are available.

wsmean The average wind speed in Germany for the corresponding time.

Details

The dataframe `data_wind_de` contains 14975 (365 x 41 + 10) rows, containing the daily average wind speed in Germany for 41 years between 1979 and 2019. Ten leap years occur within this period.

This corresponds to a subset of the data that is publicly available at https://cds.climate.copernicus.eu/cdsapp#!/dataset/reanaly era5-pressure-levels?tab=overview. Users are referred to the reference below for further details.

References

Hersbach, H et al. (2023): ERA5 hourly data on single levels from 1940 to present. Copernicus Climate Change Service (C3S) Climate Data Store (CDS) doi:10.24381/cds.adbb2d47 Accessed 01-09-2022.

Examples

```
data("wind_de")
```

`fit_dist` *Fit a distribution to data*

Description

Function to fit a specified distribution a vector of data. Returns the estimated distribution and relevant goodness-of-fit statistics.

Usage

```
fit_dist(data, dist, n_thres = 20)
```

Arguments

`data`	vector of data
`dist`	character string specifying the distribution, see details
`n_thres`	number of data points required to estimate the distribution

Details

This has been adapted from code available at `https://github.com/WillemMaetens/standaRdized`.

`data` is a numeric vector of data from which the distribution is to be estimated.

`dist` is the specified distribution to be fit to `data`. This must be one of 'empirical' (the empirical distribution given `data`), 'kde' (kernel density estimation), 'norm', 'lnorm', 'logis', 'llogis', 'exp', 'gamma', and 'weibull'.

By default, `dist = "empirical"`, in which case the distribution is estimated empirically from `data`. This is only recommended if there are at least 100 values in `data`, and a warning message is returned otherwise.

n_thres is the minimum number of observations required to fit the distribution. The default is n_thres = 20. If the number of values in data is smaller than na_thres, an error is returned. This guards against over-fitting, which can result in distributions that do not generalise well out-of-sample.

Where relevant, parameter estimation is performed using maximum likelihood estimation.

Value

A list containing the estimated distribution function, its parameters, and Kolmogorov-Smirnov goodness-of-fit statistics.

Examples

```
N <- 1000
shape <- 3
rate <- 2


# gamma distribution
data <- rgamma(N, shape, rate)
out <- fit_dist(data, dist = "gamma")
hist(data, breaks = 30, probability = TRUE)
lines(seq(0, 10, 0.01), dgamma(seq(0, 10, 0.01), out$params[1], out$params[2]), col = "blue")


# weibull distribution
data <- rweibull(N, shape, 1/rate)
out <- fit_dist(data, dist = "weibull")
hist(data, breaks = 30, probability = TRUE)
lines(seq(0, 10, 0.01), dweibull(seq(0, 10, 0.01), out$params[1], out$params[2]), col = "blue")
```

`get_drought` *Get drought characteristics*

Description

Extract characteristics of droughts from a time series of values. Drought characteristics include the occurrence, intensity, magnitude, and duration of the drought.

Usage

```
get_drought(x, thresholds = c(1.28, 1.64, 1.96), exceed = TRUE, lag = FALSE)
```

Arguments

`x`	vector or xts object.
`thresholds`	numeric vector containing thresholds to use when defining droughts.
`exceed`	logical; TRUE if a drought is defined when `x` is above the thresholds, FALSE otherwise.
`lag`	logical; TRUE if the drought should end when the value changes sign.

Details

A drought is assumed to be defined as an instance when the vector `x` exceeds (if `exceed = TRUE`) or falls below (if `exceed = FALSE`) the specified thresholds in `thresholds`.

`thresholds` can be a single value, or a vector of values. In the latter case, each threshold is assumed to be a different level or intensity of the drought. For example, if `thresholds = c(1, 1.5, 2)`, then a level 1 drought occurs whenever `x` exceeds 1 but is lower than 1.5, a level 2 drought occurs whenever `x` exceeds 1.5 but is lower than 2, and a level 3 drought occurs whenever `x` exceeds 2.

By default, `thresholds = c(1.28, 1.64, 1.96)`, which correspond to the 90th, 95th, and 97.5th percentiles of the standard normal distribution.

In meteorology, droughts are typically defined in terms of standardised indices, such as the standardised precipitation index (SPI). It is sometimes the case that a drought event ends not when the variable of interest no longer exceeds (or falls below) the relevant thresholds, but rather when the index changes sign. This can help to account for fluctuations around the threshold values, classing it as one long drought rather than several shorter droughts. This definition can be used by specifying `lag = TRUE`.

`get_drought()` currently does not use the time series information in the xts input, thereby assuming that the time series is complete, without missing time periods. If `x` is a vector, rather than an xts object, then this is also implicitly assumed.

The output is a dataframe containing the vector `x`, a logical vector specifying whether each value of `x` corresponds to a drought event, and the magnitude of the drought. The magnitude of the drought is only shown on the last day of the drought. This makes it easier to compute statistics about the drought magnitude, such as the average drought magnitude. If `thresholds` is a vector, the intensity or level of the drought is also returned.

Value

A data frame containing the original values `x` and the corresponding drought characteristics.

Author(s)

Sam Allen, Noelia Otero

References

Allen, S. and N. Otero (2023): ‘Standardised indices to monitor energy droughts’, *Renewable Energy* doi:10.1016/j.renene.2023.119206

McKee, T. B., Doesken, N. J., & Kleist, J. (1993): ‘The relationship of drought frequency and duration to time scales’, *In Proceedings of the 8th Conference on Applied Climatology* 17, 179-183.

Examples

```
data(data_supply)
# consider hourly German energy supply data in 2019
supply_de <- subset(data_supply, country == "Germany", select = c("date", "PWS"))
supply_de <- xts::xts(supply_de$PWS, order.by = supply_de$date)
supply_de_std <- std_index(supply_de, timescale = "hours")

get_drought(supply_de_std, thresholds = c(-1, -1.5, -2), exceed = FALSE)
```

get_pit — *Calculate probability integral transform values*

Description

Function to estimate the cumulative distribution function (CDF) from a set of observations, and return the corresponding probability integral transform (PIT) values.

Usage

```
get_pit(ref_data, new_data, dist = "empirical", return_fit = FALSE)
```

Arguments

`ref_data`	numeric vector from which to estimate the CDF.
`new_data`	numeric vector from which to calculate the PIT values.
`dist`	string; distribution used to estimate the CDF.
`return_fit`	logical; return parameters and goodness-of-fit statistics.

Details

`dist` specifies the distribution used to estimate the cumulative distribution function of the observations. By default, `dist = "empirical"`, in which case the CDF is estimated empirically from the values `ref_data`. This is only recommended if there are at least 100 values in `ref_data`, and a warning message is returned otherwise.

Parametric distributions are more appropriate when there is relatively little data, and good reason to expect that the data follows a particular distribution. To check that the chosen parametric distribution is appropriate, the argument `return_fit` can be used to return the estimated parameters of the distribution, as well as Kolmogorov-Smirnov goodness-of-fit test statistics.

A flexible compromise between using empirical methods and parametric distributions is to use kernel density estimation, `dist = "kde"`.

`dist` must be one of: 'empirical' (the empirical distribution given `data`), 'kde' (kernel density estimation), norm', 'lnorm', 'logis', 'llogis', 'exp', 'gamma', and 'weibull'. For the parametric distributions, parameters are estimated using maximum likelihood estimation.

Value

A vector of PIT values if return_fit = F, or, if return_fit = T, a list containing the estimated CDF (`F_x`), the corresponding parameters (`params`), and properties of the fit (`fit_props`).

Author(s)

Sam Allen, Noelia Otero

Examples

```
N <- 1000
shape <- 3
rate <- 2

x_ref <- rgamma(N, shape, rate)
x_new <- rgamma(N, shape, rate)

# empirical distribution
pit <- get_pit(x_ref, x_new)
hist(pit)

# gamma distribution
pit <- get_pit(x_ref, x_new, dist = "gamma", return_fit = TRUE)
hist(pit$pit)

hist(x_ref, breaks = 30, probability = TRUE)
lines(seq(0, 10, 0.01), dgamma(seq(0, 10, 0.01), pit$params[1], pit$params[2]), col = "blue")


# weibull distribution
pit <- get_pit(x_ref, x_new, dist = "weibull", return_fit = TRUE)
hist(pit$pit)

hist(x_ref, breaks = 30, probability = TRUE)
lines(seq(0, 10, 0.01), dweibull(seq(0, 10, 0.01), pit$params[1], pit$params[2]), col = "blue")


# exponential distribution
pit <- get_pit(x_ref, x_new, dist = "exp", return_fit = TRUE)
hist(pit$pit)

hist(x_ref, breaks = 30, probability = TRUE)
lines(seq(0, 10, 0.01), dexp(seq(0, 10, 0.01), pit$params[1]), col = "blue")
```

`plot_sei` *Plot standardised indices*

Description

Plot a time series containing standardised indices, or a histogram of the indices.

Usage

```
plot_sei(
  x,
  type = c("ts", "hist", "bar"),
  title = NULL,
  lab = "Std. Index",
  xlims = NULL,
  ylims = NULL,
  n_bins = 30
)
```

Arguments

`x`	vector or xts object containing the indices to be plotted.
`type`	type of plot (either time series "ts", histogram "hist", or barplot "bar").
`title`	optional title of the plot.
`lab`	axis label.
`xlims, ylims`	lower and upper limits of the axes.
`n_bins`	the number of bins to show in the histogram.

Details

The `plot_sei()` function can be used to plot either a time series (if `type = "ts"`) or a histogram (if `type = "hist"` or `type = "bar"`) of the values in `x`.

A time series can only be displayed if `x` is an **xts** time series.

The argument `lab` is a string containing the label of the x-axis if `type = "hist"` or `type = "bar"` and the y-axis if `type = "ts"`.

The options `type = "hist"` and `type = "bar"` both display histograms of the data `x`. With `type = "hist"`, `plot_sei()` is essentially a wrapper of `geom_histogram()`, while `type = "bar"` is a wrapper of `geom_bar()`. The latter can provide more flexibility when plotting bounded data, whereas the former is easier to use when superimposing densities on top.

Value

A ggplot object displaying the standardised index values.

Author(s)

Sam Allen, Noelia Otero

Examples

```
data(data_supply)
# consider hourly German energy supply data in 2019
supply_de <- subset(data_supply, country == "Germany", select = c("date", "PWS"))
supply_de <- xts::xts(supply_de$PWS, order.by = supply_de$date)
supply_de_std <- std_index(supply_de, timescale = "hours")

plot_sei(supply_de, title = "German renewable energy production in 2019")
plot_sei(supply_de_std, title = "German SREPI in 2019")

plot_sei(supply_de, type = "hist", title = "German renewable energy production in 2019")
plot_sei(supply_de_std, type = "hist", title = "German SREPI in 2019")
```

`std_index` *Calculate standardised indices*

Description

Inputs a time series of a chosen variable (e.g. precipitation, energy demand, residual load etc.) and returns a time series of standardised indices. Indices can be calculated on any timescale.

Usage

```
std_index(
  x_new,
  x_ref = x_new,
  timescale = NULL,
  dist = "empirical",
  return_fit = FALSE,
  moving_window = NULL,
  window_scale = NULL,
  agg_period = NULL,
  agg_scale = NULL,
  agg_fun = "sum",
  rescale = NULL,
  rescale_fun = "sum",
  index_type = "normal",
  ignore_na = FALSE
)
```

Arguments

`x_new`	numeric; vector or time series to be converted to standardised indices.
`x_ref`	numeric; vector or time series containing reference data to use when calculating the standardised indices.

`timescale`	string; timescale of the data. Required if the time series is to be aggregated or rescaled.
`dist`	string; distribution used to calculate the indices.
`return_fit`	logical; return parameters and goodness-of-fit statistics for the distribution fit.
`moving_window`	numeric; length of moving window on which to calculate the indices.
`window_scale`	string; timescale of `moving_window`, default is the timescale of the data.
`agg_period`	numeric; the number of values to aggregate over.
`agg_scale`	string; timescale of `agg_period`, default is the timescale of the data.
`agg_fun`	string; function used to aggregate the data over the aggregation period, default is "sum".
`rescale`	string; the timescale that the time series should be rescaled to.
`rescale_fun`	string; function used to rescale the data, default is "sum".
`index_type`	string; the type of index: "normal" (default), "probability", or "bounded".
`ignore_na`	logical; ignore NAs when rescaling the time series.

Details

Standardised indices are calculated by estimating the cumulative distribution function (CDF) of the variable of interest, and using this to transform the measurements to a standardised scale.

`std_index()` estimates the CDF using a time series of reference data `x_ref`, and applies the resulting transformation to the time series `x_new`. The result is a time series of standardised `x_new` values. These standardised indices quantify how extreme the `x_new` values are in reference to `x_ref`. `x_new` and `x_ref` should therefore contain values of the same variable. If `x_ref` is not specified, then `x_new` is also used to estimate the CDF.

`x_new` and `x_ref` can either be provided as vectors or xts time series. In the latter case, the time series can be aggregated across timescales or rescaled. This is useful, for example, if `x_new` contains hourly data, but interest is on daily accumulations or averages of the hourly data.

The argument `rescale` converts the data to a different timescale. The original timescale of the data can be manually specified using the argument `timescale`. Otherwise, the function will try to automatically determine the timescale of the data. Manually specifying the timescale of the data is generally more robust. The rescaling is performed using the function `rescale_fun`. By default, this is assumed to be `rescale_fun = "sum"`, so that values are added across the timescale of interest. This can be changed to any user-specified function.

The argument `agg_period` aggregates the data across the timescale of interest. This differs from `rescale` in that the resolution of the data remains the same. `agg_period` is a number specifying how long the data should be aggregated across. By default, it is assumed that `agg_period` is on the same timescale as `x_new` and `x_ref`. For example, if the data is hourly and `agg_period = 24`, then this assumes the data is to be aggregated over the past 24 hours. The scale of the aggregation period can also be specified manually using `agg_scale`. For example, one could also specify `agg_period = 1` and `agg_scale = "days"`, and this would also aggregate the data over the past day. `agg_fun` specifies how the data is to be aggregated, the default is `agg_fun = "sum"`.

`timescale`, `agg_scale`, and `rescale` must all be one of: "days", "weeks", "months", "quarters", and "years".

`dist` is the distribution used to estimate the CDF from `x_ref`. Currently, functionality is available to fit one of the following distributions to the data: Normal ('norm'), Log-normal ('lnorm'), Logistic ('logis'), Log-logistic ('llogis'), Exponential ('exp'), Gamma ('gamma'), and Weibull ('weibull'). Alternatively, the CDF can be estimated empirically (`dist = "empirical"`) based on the values in `x_ref`, or using kernel density estimation (`dist = "kde"`).

If `dist` is a parametric family of distributions, then parameters of the distribution are estimated using maximum likelihood estimation from `x_ref`. The resulting parameters and corresponding goodness-of-fit statistics can be returned by specifying `return_fit = TRUE`.

By default, the distribution is estimated over all values in `x_ref`. Alternatively, if `x_new` is an xts object, parameters can be estimated sequentially using a moving window of values. `moving_window` determines the length of the moving window. This is a single value, assumed to be on the same timescale as `x_new`. This can also be specified manually using `window_scale`. `window_scale` must also be one of "days", "weeks", "months", "quarters", and "years".

The function returns a vector of time series (depending on the format of `x_new`) containing the standardised indices corresponding to `x_new`. Three different types of indices are available, which are explained in detail in the vignette. The index type can be chosen using `index_type`, which must be one of "normal" (default), "probability", and "bounded".

Value

Time series of standardised indices.

Author(s)

Sam Allen, Noelia Otero

References

Allen, S. and N. Otero (2023): 'Standardised indices to monitor energy droughts', *Renewable Energy* 217, 119206 doi:10.1016/j.renene.2023.119206

McKee, T. B., Doesken, N. J., & Kleist, J. (1993): 'The relationship of drought frequency and duration to time scales', *In Proceedings of the 8th Conference on Applied Climatology* 17, 179-183.

Examples

```
data(data_supply)
# consider hourly German energy supply data in 2019
supply_de <- subset(data_supply, country == "Germany", select = c("date", "PWS"))
supply_de <- xts::xts(supply_de$PWS, order.by = supply_de$date)
#options(xts_check_TZ = FALSE)

# convert to hourly standardised indices
supply_de_std <- std_index(supply_de, timescale = "hours")
hist(supply_de, main = "Raw values")
hist(supply_de_std, main = "Standardised values")

# convert to daily or weekly standardised indices
supply_de_std <- std_index(supply_de, timescale = "hours", rescale = "days")
```

```
# convert to weekly standardised indices calculated on each day
supply_de_std <- std_index(supply_de, timescale = "hours", rescale = "days",
                           agg_period = 1, agg_scale = "weeks")

# calculate standardised indices corresponding to December, based on the previous year
dec <- zoo::index(supply_de) > "2019-12-01 UTC"
supply_de_std_dec <- std_index(x_new = supply_de[dec], x_ref = supply_de[!dec],
                               timescale = "hours")

# calculate standardised indices using a 100 day moving window
supply_de_std_dec <- std_index(supply_de[dec], supply_de, timescale = "hours",
                               rescale = "days", moving_window = 100)

# suppose we are interested in the daily maximum rather than the daily total
supply_de_std <- std_index(supply_de, timescale = "hours", rescale = "days",
                           rescale_fun = "max")
supply_de_std <- std_index(supply_de, timescale = "hours", rescale = "days",
                           rescale_fun = "mean") # or average

# the default uses the empirical distribution, but this requires more data than
# parametric distributions, meaning it is not ideal when data is short, e.g. in weekly case
supply_de_std <- std_index(supply_de, timescale = "hours", rescale = "weeks") # warning
# instead, we can use a parametric distribution, e.g. a gamma distribution
supply_de_std <- std_index(supply_de, timescale = "hours", rescale = "weeks", dist = "gamma")
# we can check the fit by checking whether the indices resemble a standard normal distribution
hist(supply_de)
hist(supply_de_std)
# we can also look at the properties of the fit
supply_de_std <- std_index(supply_de, timescale = "hours", rescale = "weeks",
                           dist = "gamma", return_fit = TRUE)

# we could also use kernel density estimation, which is a flexible compromise between the two
supply_de_std <- std_index(supply_de, timescale = "hours", rescale = "weeks", dist = "kde")
```

Index