

# Package ‘SEI’

May 7, 2026

**Title** Calculating Standardised Indices

**Version** 0.2.0

**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-parametric and non-stationary methods.

**URL** <https://github.com/noeliaof/SEI>, <https://noeliaof.github.io/SEI/>

**License** GPL (>= 2)

**Encoding** UTF-8

**RoxygenNote** 7.2.3

**Depends** R (>= 3.5.0)

**Imports** ggplot2, xts, zoo, fitdistrplus, flexsurv, gamlss,  
gamlss.dist, lmom

**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@stat.math.ethz.ch>

**Repository** CRAN

**Date/Publication** 2024-08-27 11:20:46 UTC

## Contents

aggregate_xts . . . . .	2
data_supply . . . . .	4
data_wind_de . . . . .	5

fit_dist . . . . .	6
get_drought . . . . .	9
get_pit . . . . .	12
plot_sei . . . . .	16
std_index . . . . .	18

<b>Index</b>	<b>24</b>
--------------	-----------

---

aggregate_xts	<i>Aggregate values in xts objects</i>
---------------	--

---

## 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,
  agg_period = 1,
  agg_scale = c("days", "mins", "hours", "weeks", "months", "years"),
  agg_fun = "sum",
  timescale = c("days", "mins", "hours", "weeks", "months", "years"),
  na_thres = 10
)
```

## Arguments

x	xts object to be aggregated.
agg_period	length of the aggregation period.
agg_scale	timescale of agg_period; one of 'mins', 'hours', 'days', 'weeks', 'months', 'years'.
agg_fun	string specifying the function used to aggregate the data over the aggregation period, default is 'sum'.
timescale	timescale of the data; one of 'mins', 'hours', 'days', 'weeks', 'months', 'years'.
na_thres	threshold for the percentage of NA values allowed in the aggregation period; default is 10%.

## Details

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

Given a vector  $x_1, x_2, \dots$ , the function `aggregate_xts` calculates aggregated values  $\tilde{x}_1, \tilde{x}_2, \dots$  as

$$\tilde{x}_t = f(x_t, x_{t-1}, \dots, x_{t-k+1}),$$

for each time point  $t = k, k+1, \dots$ , where  $k$  (`agg_period`) is the number of time units (`agg_scale`) over which to aggregate the time series ( $x$ ), and  $f$  (`agg_fun`) is the function used to perform the aggregation. The first  $k - 1$  values of the aggregated time series are returned as NA.

By default, `agg_fun = "sum"`, meaning the aggregation results in accumulations over the aggregation period:

$$\tilde{x}_t = \sum_{k=1}^K x_{t-k+1}.$$

Alternative functions can also be used. For example, specifying `agg_fun = "mean"` returns the mean over the aggregation period,

$$\tilde{x}_t = \frac{1}{K} \sum_{k=1}^K x_{t-k+1},$$

while `agg_fun = "max"` returns the maximum over the aggregation period,

$$\tilde{x}_t = \max(\{x_t, x_{t-1}, \dots, x_{t-k+1}\}).$$

`agg_period` is a single numeric value specifying over how many time units the data  $x$  is to be aggregated. By default, `agg_period` is assumed to correspond to a number of days, but this can also be specified manually using the argument `agg_scale`. `timescale` is the timescale of the input data  $x$ . By default, this is also 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, timescale = "hours")

# weekly means
supply_de_weekly <- aggregate_xts(supply_de, agg_scale = "weeks",
```

```
agg_fun = "mean", timescale = "hours")

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

---

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](https://doi.org/10.17864/1947.000321)

## Examples

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

---

data_wind_de	<i>Time series of average wind speed in Germany</i>
--------------	---

---

## 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 ( $365 \times 41 + 10$ ) 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/reanalysis-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](https://doi.org/10.24381/cds.adbb2d47) Accessed 01-09-2022.

## Examples

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

---

fit_dist	<i>Fit a distribution to data</i>
----------	-----------------------------------

---

### Description

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

### Usage

```
fit_dist(data, dist, method = "mle", preds = NULL, n_thres = 10, ...)
```

### Arguments

data	A numeric vector.
dist	character string specifying the distribution to be fit to the data; one of 'empirical', 'kde', 'norm', 'lnorm', 'logis', 'llogis', 'exp', 'gamma', and 'weibull'.
method	A character string coding for the fitting method: "mle" for 'maximum likelihood estimation', "mme" for 'moment matching estimation', "qme" for 'quantile matching estimation', "mge" for 'maximum goodness-of-fit estimation' and "mse" for 'maximum spacing estimation'.
preds	data frame of predictor variables on which the estimated distribution should depend.
n_thres	minimum number of data points required to estimate the distribution; default is 10.
...	additional arguments to be passed to <a href="#">fitdist</a> or <a href="#">gamlss</a>

### 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', 'kde', 'norm', 'lnorm', 'logis', 'llogis', 'exp', 'gamma', and 'weibull'. These correspond to the following distributions: 'empirical' returns the empirical distribution function of data, 'kde' applies (normal) kernel density estimation to data, while 'norm', 'lnorm', 'logis', 'llogis', 'exp', 'gamma', and 'weibull' correspond to the normal, log-normal, logistic, log-logistic, exponential, gamma, and Weibull distributions, respectively.

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. Parametric distributions are more appropriate when there is relatively little data, or good reason to expect that the data follows a particular distribution. Kernel density estimation dist = 'kde' provides a flexible compromise between using empirical methods and parametric distributions.

n\_thres is the minimum number of observations required to fit the distribution. The default is n\_thres = 10. 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.

method specifies the method used to estimate the distribution parameters. This argument is redundant if `dist = 'empirical'` or `dist = 'kde'`. Otherwise, `fit_dist` essentially provides a wrapper for `fitdist`, and further details can be found in the corresponding documentation. Additional arguments to `fitdist` can also be specified via `...`. Where relevant, the default is to estimate parameters using maximum likelihood estimation, `method = "mle"`, though several alternative methods are also available; see `fitdist`. Parameter estimation is also possible using L-moment matching (`method = 'lmme'`), for all distribution choices except the log-logistic distribution. In this case, `fit_dist` is essentially a wrapper for the `lmom` package.

The distribution can also be non-stationary, by depending on some predictor variables or covariates. These predictors can be included via the argument `preds`, which should be a data frame with a separate column for each predictor, and with a number of rows equal to the length of data. In this case, a Generalized Additive Model for Location, Scale, and Shape (GAMLSS) is fit to data using the predictors in `preds`. It is assumed that the mean of the distribution depends linearly on all of the predictors. Variable arguments in `...` can also be used to specify relationships between the scale and shape parameters of the distribution and the predictors; see examples below. In this case, `fit_dist` is essentially a wrapper for `gamlss`, and users are referred to the corresponding documentation for further implementation details.

### Value

A list containing the estimated distribution function (`F_x`), its parameters (`params`), and properties of the fit such as the AIC and Kolmogorov-Smirnov goodness-of-fit statistic (`fit`). If the estimated distribution function depends on covariates, then the `gamlss` model fit is returned as the parameters.

### Author(s)

Sam Allen, Noelia Otero

### References

- Rigby, R. A., & Stasinopoulos, D. M. (2005): 'Generalized additive models for location, scale and shape', *Journal of the Royal Statistical Society Series C: Applied Statistics* 54, 507-554. doi:10.1111/j.14679876.2005.00510.x
- Delignette-Muller, M. L., & Dutang, C. (2015): 'fitdistrplus: An R package for fitting distributions', *Journal of Statistical Software* 64, 1-34. doi:10.18637/jss.v064.i04
- Allen, S. & N. Otero (2023): 'Standardised indices to monitor energy droughts', *Renewable Energy* 217, 119206. doi:10.1016/j.renene.2023.119206

### See Also

[fitdist](#) [gamlss](#) [lmom](#)

### Examples

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

```
x <- seq(0, 10, 0.01)

### gamma distribution

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

# method of moments
out <- fit_dist(data, dist = "gamma", method = "mme")
hist(data, breaks = 30, probability = TRUE)
lines(x, dgamma(x, out$params[1], out$params[2]), col = "blue")

# method of l-moments
out <- fit_dist(data, dist = "gamma", method = "lmme")
hist(data, breaks = 30, probability = TRUE)
lines(x, dgamma(x, out$params[1], out$params[2]), col = "blue")

## weibull distribution

# maximum likelihood
data <- rweibull(N, shape, 1/rate)
out <- fit_dist(data, dist = "weibull")
hist(data, breaks = 30, probability = TRUE)
lines(x, dweibull(x, out$params[1], out$params[2]), col = "blue")

# method of l-moments
out <- fit_dist(data, dist = "weibull", method = "lmme")
hist(data, breaks = 30, probability = TRUE)
lines(x, dweibull(x, out$params[1], out$params[2]), col = "blue")

## exponential distribution

# method of moments
out <- fit_dist(data, dist = "exp", method = "mme")
hist(data, breaks = 30, probability = TRUE)
lines(x, dexp(x, out$params), col = "blue")

## logistic distribution

x <- seq(-10, 20, 0.01)

# maximum likelihood
data <- rlogis(N, shape, rate)
out <- fit_dist(data, dist = "logis")
hist(data, breaks = 30, probability = TRUE)
lines(x, dlogis(x, out$params[1], out$params[2]), col = "blue")
```

```
##### non-stationary estimation using gamlss

## normal distribution
x <- seq(-10, 20, length.out = N)
data <- rnorm(N, x + shape, exp(x/10))
plot(data)
preds <- data.frame(t = x)

out_st <- fit_dist(data, dist = "norm")
out_nst <- fit_dist(data, dist = "norm", preds = preds)
out_nst2 <- fit_dist(data, dist = "norm", preds = preds, sigma.formula = ~ .)

# pit values without trend
pit_st <- out_st$F_x(data, out_st$params)
hist(pit_st, breaks = 30, probability = TRUE, main = "No trend")
abline(1, 0, col = "red", lty = "dotted")
# pit values with trend in mean
pit_nst <- out_nst$F_x(data, out_nst$params, preds)
hist(pit_nst, breaks = 30, probability = TRUE, main = "Trend in mean")
abline(1, 0, col = "red", lty = "dotted")
# pit values with trend in mean and sd
pit_nst2 <- out_nst2$F_x(data, out_nst2$params, preds)
hist(pit_nst2, breaks = 30, probability = TRUE, main = "Trend in mean and standard deviation")
abline(1, 0, col = "red", lty = "dotted")

## log normal distribution
x <- seq(0.01, 10, length.out = N)
data <- rlnorm(N, (x + shape)/3, 1/rate)
plot(data)
preds <- data.frame(t = x)

out <- fit_dist(data, dist = "lnorm", preds = preds)
pit <- out$F_x(data, out$params, preds)
hist(pit, breaks = 30, probability = TRUE, main = "PIT values for non-stationary fit")
abline(1, 0, col = "red", lty = "dotted")
```

---

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,
  cluster = 0,
  lag = NULL
)
```

**Arguments**

<code>x</code>	vector or xts object from which droughts are defined.
<code>thresholds</code>	numeric vector containing thresholds to use when defining droughts.
<code>exceed</code>	logical; TRUE if a drought is defined when <code>x</code> is above the thresholds, FALSE otherwise.
<code>cluster</code>	integer specifying the number of time steps over which droughts should be clustered.
<code>lag</code>	numeric specifying the value at which the drought should end.

**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. If `exceed = TRUE` then a higher threshold corresponds to a higher intensity, and if `exceed = FALSE` then a lower threshold corresponds to a higher intensity. 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 corresponds to the 90th, 95th, and 97.5th percentiles of the standard normal distribution. These thresholds are often used alongside standardised indices to define hydrometeorological droughts; see references.

`cluster` represents the number of time steps between different drought events that should be attributed to the same drought. For example, suppose  $x_i \geq t, x_{i+1} < t, x_{i+2} \geq t$ , where  $x_i$  represents the  $i$ -th value in `x`, and  $t$  is the lowest threshold in `thresholds`. In this case, one drought event will finish at time point  $i$  and a new drought event will begin at time point  $i + 2$ ; no drought will occur at time point  $i + 1$  because the value  $x_{i+1}$  is below the threshold defining a drought. Since both  $x_i$  and  $x_{i+2}$  are classed as drought events, it may be desirable to ignore the fluctuation, and assume that the drought persists through  $x_{i+1}$  despite its value. This can be achieved by setting `cluster = 1`. If there were two time points separating different drought events, these can be clustered together by setting `cluster = 2`, and so on. The default is that no clustering should be implemented, i.e. `cluster = 0`.

Alternatively, we may wish to assume that the drought persists until `x` falls below a value that is not necessarily equal to the threshold defining a drought. For example, hydrometeorological droughts based on standardised indices, such as the Standardised Precipitation Index (SPI), are often defined to persist until the standardised index changes sign, i.e. falls below zero. This can be achieved by

setting `lag = 0`. More generally, `lag` can be any numerical value. If `exceed = TRUE`, a warning is issued if `lag` is above the lowest threshold, and if `exceed = FALSE`, a warning is issued if `lag` is below the highest threshold. If `lag` is `NULL` (the default), then no lagging is performed.

`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, defined as the sum of the values of `x` during the drought; see references. 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

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.

Vicente-Serrano, S. M., Beguería, S., & López-Moreno, J. I. (2010): 'A multiscalar drought index sensitive to global warming: the standardized precipitation evapotranspiration index', *Journal of Climate* 23, 1696-1718. doi:[10.1175/2009JCLI2909.1](https://doi.org/10.1175/2009JCLI2909.1)

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

### Examples

```
data(data_supply)

# consider daily 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, rescale = "days", timescale = "hours")

# a drought may correspond to when energy supply is low
drought_df <- get_drought(supply_de_std, thresholds = c(-1.28, -1.64, -1.96), exceed = FALSE)
head(drought_df)
mean(drought_df$occ) # droughts occur on roughly 10% of time steps

# cluster droughts two time steps apart
drought_df <- get_drought(supply_de_std, thresholds = c(-1.28, -1.64, -1.96),
  cluster = 2, exceed = FALSE)
mean(drought_df$occ) # droughts occur on roughly 11% of time steps
```

```
# let droughts persist until the standardised index changes sign
drought_df <- get_drought(supply_de_std, thresholds = c(-1.28, -1.64, -1.96),
                        lag = 0, exceed = FALSE)
mean(drought_df$occ) # droughts occur on roughly 17% of time steps
```

---

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(
  x_ref,
  x_new = x_ref,
  dist = "empirical",
  preds_ref = NULL,
  preds_new = preds_ref,
  method = "mle",
  return_fit = FALSE,
  lower = -Inf,
  upper = Inf,
  cens = "none",
  n_thres = 10,
  ...
)
```

### Arguments

x_ref	numeric vector from which to estimate the CDF.
x_new	numeric vector from which to calculate the PIT values.
dist	character string specifying the distribution to be fit to the data; one of 'empirical', 'kde', 'norm', 'lnorm', 'logis', 'llogis', 'exp', 'gamma', and 'weibull'.
preds_ref	data frame of predictor variables on which the estimated distribution should depend, corresponding to the reference observations x_ref.
preds_new	data frame of predictor variables on which the estimated distribution should depend, corresponding to the new observations x_new.
method	A character string coding for the fitting method: "mle" for 'maximum likelihood estimation', "mme" for 'moment matching estimation', "qme" for 'quantile matching estimation', "mge" for 'maximum goodness-of-fit estimation' and "mse" for 'maximum spacing estimation'.

return_fit	logical specifying whether to return parameters and goodness-of-fit statistics for the distribution fit.
lower, upper	numeric values specifying the lower and upper bounds at which the values in x_ref and x_new are censored.
cens	method used to deal with censoring of the PIT values; either a string ('none', 'normal' or 'prob'), corresponding to common choices, or a custom numeric value.
n_thres	minimum number of data points required to estimate the distribution; default is 10.
...	additional arguments to be passed to <a href="#">fitdist</a> or <a href="#">gamlss</a>

## Details

### Continuous data

If  $X$  is a continuous random variable with cumulative distribution function (CDF)  $F$ , then the probability integral transform (PIT)  $F(X)$  is uniformly distributed between 0 and 1. Given a vector  $x_1, \dots, x_n$  of realisations of  $X$ , `get_pit` produces an estimate  $\hat{F}$  of the CDF  $F$ , and returns a vector of PIT values corresponding to another set of realisations  $z_1, \dots, z_N$ ,

$$\hat{F}(z_1), \dots, \hat{F}(z_n).$$

`x_ref` contains the values  $x_1, \dots, x_n$  from which the CDF estimate  $\hat{F}$  is obtained. `x_new` contains the values  $z_1, \dots, z_n$  from which the PIT values  $\hat{F}(z_1), \dots, \hat{F}(z_n)$  are calculated. By default, `x_ref` and `x_new` are the same, so that the PIT values are calculated in-sample.

To estimate the distribution, `get_pit` calls `fit_dist`. The arguments `dist`, `method` and `n_thres` are documented in detail in the corresponding help page.

To check that the chosen distribution adequately fits the data, the argument `return_fit = TRUE` can be used to return the estimated parameters of the distribution, as well as properties of the fit such as the AIC and a p-value for the Kolmogorov-Smirnov goodness-of-fit test.

### Non-stationary distributions

The estimated distribution can also be non-stationary, by depending on some predictor variables or covariates. These predictors can be included via the arguments `preds_ref` and `preds_new`, which should be data frames with a separate column for each predictor, and with numbers of rows equal to the lengths of `x_ref` and `x_new`, respectively. In this case, a Generalized Additive Model for Location, Scale, and Shape (GAMLSS) is fit to `x_ref` using the predictors in `preds_ref`. The PIT values corresponding to `x_new` are then calculated by applying the estimated distribution with predictors `preds_new`. If a non-stationary distribution is to be estimated, both `preds_ref` and `preds_new` must be provided. By default, `preds_new` is assumed to be the same as `preds_ref`, to align with `x_new` being the same as `x_ref`.

### Censored data

If the random variable  $X$  is not continuous, the PIT will not be uniformly distributed. A relevant case is when  $X$  is censored. For example, precipitation is censored below at zero. This results in several PIT values being equal to  $F(0)$ . The lower and upper arguments to `get_pit` allow the user to specify the lower and upper bounds at which the data is censored; the default is `lower = -Inf` and `upper = Inf`, i.e. there is no censoring.

If the PIT values are used to construct standardised indices, this censoring can lead to unintuitive index values. To deal with censored data, it has been proposed to map the PIT values of the censored values to a different constant  $c$ ; see references. For example, for precipitation, the PIT values would become

$$F(X) \quad \text{if } X > 0,$$

$$c \quad \text{if } X = 0.$$

The constant  $c$  can be chosen so that the PIT values satisfy some desired property. For example, if  $F(X)$  is uniformly distributed between 0 and 1, then it has mean equal to  $1/2$ . Hence,  $c$  could be chosen such that the mean of the PIT values of the censored distribution are equal to  $1/2$ . Alternatively, if  $F(X)$  is uniformly distributed between 0 and 1, then the transformed PIT value  $\Phi^{-1}(F(X))$  (where  $\Phi^{-1}$  is the quantile function of the standard normal distribution) follows a standard normal distribution, and therefore has mean equal to 0. The constant  $c$  could therefore be chosen such that the mean of the transformed PIT values of the censored distribution are equal to 0.

The argument `cens` in `get_pit` can be used to treat censored data. `cens` can be one of four options: a single numeric value containing the value  $c$  at which to assign the PIT values of the censored realisations; the string 'none' if no censoring is to be performed; the string 'prob' if  $c$  is to be chosen automatically so that the mean of the PIT values is equal to  $1/2$ ; or the string 'normal' if  $c$  is to be chosen automatically so that the mean of the transformed PIT values is equal to 0. If the data is censored both above and below, then `cens` must be a numeric vector of length two, specifying the values to assign the realisations that are censored both below and above.

When the data is censored, `dist` corresponds to the distribution used to estimate the uncensored realisations, e.g. positive precipitations. The probability of being at the boundary points is estimated using the relative frequency of censored observations in `x_ref`.

## Value

A vector of PIT values if `return_fit = FALSE`, or, if `return_fit = TRUE`, a list containing the estimated distribution function (`F_x`), its parameters (`params`), and properties of the fit such as the AIC and Kolmogorov-Smirnov goodness-of-fit statistic (`fit`). If the estimated distribution function depends on covariates, then the `gamLSS` model fit is returned as the parameters.

## Author(s)

Sam Allen, Noelia Otero

## References

- Rigby, R. A., & Stasinopoulos, D. M. (2005): 'Generalized additive models for location, scale and shape', *Journal of the Royal Statistical Society Series C: Applied Statistics* 54, 507-554. doi:10.1111/j.14679876.2005.00510.x
- Stagge, J. H., Tallaksen, L. M., Gudmundsson, L., Van Loon, A. F., & Stahl, K. (2015): 'Candidate distributions for climatological drought indices (SPI and SPEI)', *International Journal of Climatology* 35, 4027-4040. doi:10.1002/joc.4267
- Allen, S. & N. Otero (2023): 'Calculating standardised indices using SEI', *EarthArXiv pre-print*. doi:10.31223/X5GM4G

**See Also**[fit\\_dist](#)**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")

# gamma distribution with censoring
x_ref <- c(x_ref, numeric(N))
pit <- get_pit(x_ref, dist = "gamma", lower = 0, cens = "prob")
hist(pit)
mean(pit) # = 1/2
mean(qnorm(pit)) # != 0

pit <- get_pit(x_ref, dist = "gamma", lower = 0, cens = "normal")
hist(qnorm(pit))
mean(pit) # != 1/2
mean(qnorm(pit)) # = 0
```

```

## normal distribution with trend in mean
x <- seq(-10, 20, length.out = N)
x_ref <- rnorm(N, x + shape, 2)
plot(x_ref)
preds <- data.frame(t = x)

pit <- get_pit(x_ref, preds_ref = preds, dist = "norm")
hist(pit)

## normal distribution with trend in mean and standard deviation
x_ref <- rnorm(N, x + shape, exp(x/10))
plot(x_ref)
preds <- data.frame(t = x)

pit <- get_pit(x_ref, preds_ref = preds, dist = "norm", sigma.formula = ~ .)
hist(pit)
# sigma.formula is an optional argument in the gamlss::gamlss function

```

---

plot\_sei

*Plot standardised indices*


---

## Description

Plot a time series or histogram of standardised 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")

# type = "hist" and type = "bar" both output a histogram of the index values
# type = "hist" can be useful to superimpose densities on top of the histogram
z <- seq(-3.5, 3.5, length.out = length(supply_de_std))
plot_sei(supply_de_std, type = "hist", title = "German SREPI in 2019") +
  ggplot2::geom_line(ggplot2::aes(x = z, y = dnorm(z)), col = "blue")

# type = "bar" can be useful when the index values are bounded
supply_de_std <- std_index(supply_de, timescale = "hours", index_type = "prob11")
plot_sei(supply_de_std, type = "hist", xlims = c(-1, 1), title = 'type = "hist"')
plot_sei(supply_de_std, type = "bar", xlims = c(-1, 1), title = 'type = "bar"')
```

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,
  dist = "empirical",
  preds_new = NULL,
  preds_ref = preds_new,
  method = "mle",
  return_fit = FALSE,
  index_type = "normal",
  gr_new = NULL,
  gr_ref = gr_new,
  timescale = NULL,
  moving_window = NULL,
  window_scale = NULL,
  agg_period = NULL,
  agg_scale = NULL,
  agg_fun = "sum",
  rescale = NULL,
  rescale_fun = "sum",
  ignore_na = FALSE,
  n_thres = 10,
  na_thres = 10,
  lower = -Inf,
  upper = Inf,
  cens = index_type,
  ...
)
```

**Arguments**

x_new	vector or time series to be converted to standardised indices.
x_ref	vector or time series containing reference data to use when calculating the standardised indices.
dist	character string specifying the distribution to be fit to the data; one of 'empirical', 'kde', 'norm', 'lnorm', 'logis', 'llogis', 'exp', 'gamma', and 'weibull'.
preds_new	data frame of predictor variables on which the estimated distribution should depend, corresponding to the new observations x_new.

preds_ref	data frame of predictor variables on which the estimated distribution should depend, corresponding to the reference observations <code>x_ref</code> .
method	A character string coding for the fitting method: "mle" for 'maximum likelihood estimation', "mme" for 'moment matching estimation', "qme" for 'quantile matching estimation', "mge" for 'maximum goodness-of-fit estimation' and "mse" for 'maximum spacing estimation'.
return_fit	logical specifying whether to return parameters and goodness-of-fit statistics for the distribution fit.
index_type	the type of standardised index: "normal" (default), "prob01", or "prob11" (see details).
gr_new	vector of factors for which separate distributions should be applied to <code>x_new</code> .
gr_ref	vector of factors for which separate distributions should be fit to <code>x_ref</code> .
timescale	timescale of the data; one of 'mins', 'hours', 'days', 'weeks', 'months', 'years'.
moving_window	length of moving window on which to calculate the indices.
window_scale	timescale of moving_window; default is the timescale of the data.
agg_period	length of the aggregation period.
agg_scale	timescale of agg_period; one of 'mins', 'hours', 'days', 'weeks', 'months', 'years'.
agg_fun	string specifying the function used to aggregate the data over the aggregation period, default is 'sum'.
rescale	the timescale that the time series should be rescaled to; one of "days", "weeks", "months", "quarters", and "years".
rescale_fun	string specifying the function used to rescale the data; default is "sum".
ignore_na	logical specifying whether to ignore NAs when rescaling the time series.
n_thres	minimum number of data points required to estimate the distribution; default is 10.
na_thres	threshold for the percentage of NA values allowed in the aggregation period; default is 10%.
lower, upper	numeric values specifying the lower and upper bounds at which the values in <code>x_ref</code> and <code>x_new</code> are censored.
cens	method used to deal with censoring of the PIT values; either a string ('none', 'normal' or 'prob'), corresponding to common choices, or a custom numeric value.
...	additional arguments to be passed to <code>fitdist</code> or <code>gamlss</code>

## Details

### Standardised indices

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 is a wrapper for [get\\_pit](#) and [fit\\_dist](#) that additionally allows for aggregation, rescaling, and grouping of the time series. Further details can be found in the help pages of [get\\_pit](#) and [fit\\_dist](#).

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 it is set equal to `x_new`, so that the standardised indices are calculated in-sample.

The function returns a vector or 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), "prob01", and "prob11".

### Time series manipulations

`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`. `timescale` is required if the time series is to be aggregated or rescaled. Otherwise, `std_index` 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, `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. The aggregation is performed using [aggregate\\_xts](#). 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, specifying `agg_period = 1` and `agg_scale = "days"` 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"`.

### Distribution estimation

`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 from `x_ref`. `method` specifies how the parameters are estimated; see [fit\\_dist](#) for details. 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`. The timescale of the moving window can also be specified manually using `window_scale`. `window_scale` must also be one of "days", "weeks", "months", "quarters", and "years".

The estimated distribution can also be non-stationary, by depending on some predictors or covariates. These predictors can be stored in data frames and input to `std_index` via the arguments `preds_new` and `preds_ref`; see [fit\\_dist](#) for details. Predictors cannot be used if the data is to be rescaled, since this would also require rescaling the predictors; in this case, an error is returned.

### Grouping

By default, one distribution is fit to all values in `x_ref`. Separate distributions can be fit to different subsets of the data by specifying `gr_ref` and `gr_new`. These should be factor vectors, where each factor corresponds to a different grouping or subset of the data. No factor should appear in `gr_new` that does not appear in `gr_ref`, since there would be no data from which to estimate the distribution for this group. An error is returned in this case. Since the distribution of the values in `x_ref` could change for different groupings, the argument `dist` can be a vector of strings of the same length as the number of factor levels in `gr_new`. In this case, the first element of `dist` should correspond to the first element of `levels(gr_new)` and so on. If `dist` is a single string, then the same distribution is used for each grouping.

### Value

Time series of standardised indices. If `return_fit = TRUE`, then a list is returned that contains the time series of standardised indices, as well as information about the fit of the distribution to the data. If `gr_new` is specified, then `std_index` returns a list of time series of standardised indices, with an element corresponding to each factor in `gr_new`.

### Author(s)

Sam Allen, Noelia Otero

### References

- 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.
- Vicente-Serrano, S. M., Beguería, S., & López-Moreno, J. I. (2010): 'A multiscalar drought index sensitive to global warming: the standardized precipitation evapotranspiration index', *Journal of Climate* 23, 1696-1718. doi:[10.1175/2009JCLI2909.1](https://doi.org/10.1175/2009JCLI2909.1)
- Allen, S. & N. Otero (2023): 'Standardised indices to monitor energy droughts', *Renewable Energy* 217, 119206. doi:[10.1016/j.renene.2023.119206](https://doi.org/10.1016/j.renene.2023.119206)

### See Also

[xts](#) [aggregate\\_xts](#) [get\\_pit](#) [fit\\_dist](#)

### 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")

# calculate separate indices for each quarter of 2019
season <- ceiling(lubridate::month(zoo::index(supply_de)) / 3)
season <- factor(c("Q1", "Q2", "Q3", "Q4")[season])
supply_de_std <- std_index(supply_de, timescale = "hours", rescale = "days",
                          gr_new = season, dist = "kde", return_fit = TRUE)

```

```
# non-stationary distribution estimation using gamlss

N <- 1000
x <- seq(-10, 20, length.out = N)
data <- rnorm(N, x, exp(x/10)) # non-stationary mean and standard deviation
plot.ts(data)
preds <- data.frame(t = x)

# standardised indices without trend
si_st <- std_index(data, dist = "norm")
plot_sei(si_st)
# standardised indices with trend in mean
si_nst <- std_index(data, dist = "norm", preds_new = preds)
plot_sei(si_nst)
# standardised indices with trend in mean and sd
si_nst2 <- std_index(data, dist = "norm", preds_new = preds, sigma.formula = ~ .)
plot_sei(si_nst2)
```

# Index

## \* datasets

data\_supply, 4  
data\_wind\_de, 5

aggregate\_xts, [2](#), [20](#), [21](#)

data\_supply, 4  
data\_wind\_de, 5

fit\_dist, [6](#), [13](#), [15](#), [20](#), [21](#)  
fitdist, [6](#), [7](#), [13](#), [19](#)

gamlss, [6](#), [7](#), [13](#), [19](#)  
get\_drought, 9  
get\_pit, [12](#), [20](#), [21](#)

lmom, 7

plot\_sei, 16

std\_index, 18

xts, [20](#), [21](#)