Package 'aedseo'

May 7, 2026

Title Automated and Early Detection of Seasonal Epidemic Onset and Burden Levels

Version 1.1.0

Description A powerful tool for automating the early detection of seasonal epidemic onsets in time series data. It offers the ability to estimate growth rates across consecutive time intervals, calculate the sum of cases (SoC) within those intervals, and estimate seasonal onsets within user defined seasons. With use of a disease-specific threshold it also offers the possibility to estimate seasonal onset of epidemics.
Additionally it offers the ability to estimate burden levels for seasons based on historical data. It is aimed towards epidemiologists, public health professionals, and researchers seeking to identify and respond to seasonal epidemics in a timely fashion.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.3

URL https://github.com/ssi-dk/aedseo, https://ssi-dk.github.io/aedseo/

BugReports https://github.com/ssi-dk/aedseo/issues

Depends R (>= 4.2.0)

Suggests grid, ISOweek, kableExtra, knitr, mem, rmarkdown, testthat (>= 3.0.0), withr

Config/testthat/edition 3

Imports base, checkmate, dplyr, ggplot2, lifecycle, lubridate, plyr, purrr, pracma, rlang, scales, stats, stringr, tibble, tidyr

Config/Needs/website rmarkdown

VignetteBuilder knitr

NeedsCompilation no

Author Sofia Myrup Otero [aut] (ORCID: <https://orcid.org/0009-0006-4953-614X>), Kasper Schou Telkamp [aut] (ORCID: <https://orcid.org/0009-0001-5126-0190>),

Lasse Engbo Christiansen [aut, cre] (ORCID:
<https://orcid.org/0000-0001-5019-1931>),
Rasmus Skytte Randløv [rev] (ORCID:
<https://orcid.org/0000-0002-5860-3838>),
Statens Serum Institut, SSI [cph, fnd]

Maintainer Lasse Engbo Christiansen <lsec@ssi.dk>

Repository CRAN

Date/Publication 2026-01-23 14:50:01 UTC

Contents

`autoplot` *Autoplot a* `tsd` *object*

Description

Generates a complete 'ggplot' object suitable for visualizing time series data in a `tsd`, `tsd_onset`, `tsd_onset_and_burden` or `tsd_growth_warning` object.

`autoplot(tsd)`

- Generates points for each observation and connects them with a line.

`autoplot(tsd_onset)`

- The first plot generates a line connecting the observations. The transparency of the points reflects if seasonal onset has occurred.
- The second plot presents the growth rate for each observation along with confidence intervals. The transparency of the points indicates whether a growth warning condition is met.

```
autoplot(tsd_onset_and_burden)
```

- Generates a line connecting the observations in the current season, along with colored regions representing different burdens levels and a vertical line indicating seasonal onset. The y-axis is scaled with `ggplot2::scale_y_log10` to give better visualisation of the burden levels.

Usage

```
autoplot(object, ...)

## S3 method for class 'tsd'
autoplot(
  object,
  line_width = 0.7,
  obs_size = 2,
  text_family = "sans",
  time_interval_step = "5 weeks",
  ...
)

## S3 method for class 'tsd_onset'
autoplot(
  object,
  disease_color = "black",
  line_width = 0.7,
  obs_size = 2,
  alpha_warning = 0.2,
  alpha_ribbon = 0.1,
  text_family = "sans",
  legend_position = "bottom",
  time_interval_step = "5 weeks",
  ...
)

## S3 method for class 'tsd_onset_and_burden'
autoplot(
  object,
  y_lower_bound = 5,
  factor_to_max = 2,
  disease_color = "#009DD1",
  season_start = 21,
  season_end = season_start - 1,
  time_interval_step = "3 weeks",
  text_burden_size = 10/2.8,
  fill_alpha = c(0.45, 0.6, 0.75, 0.89, 1),
  text_family = "sans",
  line_color = "black",
  line_type = "solid",
  vline_color_onset = "#bf212f",
```

```
  vline_linetype_onset = "dashed",
  vline_color_offset = "#006f3c",
  vline_linetype_offset = "dotted",
  line_width = 1,
  y_scale_labels = scales::label_comma(big.mark = ".", decimal.mark = ","),
  theme_custom = ggplot2::theme_bw(),
  legend_position = "right",
  ...
)

## S3 method for class 'tsd_growth_warning'
autoplot(
  object,
  k = 5,
  skip_current_season = TRUE,
  line_width = 1,
  text_family = "sans",
  legend_position = "bottom",
  breaks_y_axis = 8,
  ...
)
```

Arguments

`object`	A `tsd_growth_warning` object
`...`	Additional arguments (not used).
`line_width`	A numeric specifying the width of line connecting observations.
`obs_size`	A numeric, specifying the size of observational points.
`text_family`	A character specifying the font family for the text labels.
`time_interval_step`	A character vector specifying the time interval and how many time steps are desired on the x-axis, e.g. '10 days', '4 weeks', or '3 months'.
`disease_color`	A character specifying the base color of the disease.
`alpha_warning`	A numeric specifying the alpha (transparency) for the observations with a seasonal_onset_alarm (first plot) or significantly positive growth rate (second plot).
`alpha_ribbon`	A numeric specifying the alpha for the confidence intervals of the growth rate.
`legend_position`	A character specifying the position of the legend on the plot.
`y_lower_bound`	A numeric specifying the lower bound of the y-axis.
`factor_to_max`	A numeric specifying the factor to multiply the high burden level for extending the y-axis.
`season_start, season_end`	Integers giving the start and end weeks of the seasons to stratify the observations by.
`text_burden_size`	A numeric specifying the size of the text labels.

`fill_alpha` A numeric vector specifying the transparency levels for the fill colors of burden levels. Must match the number of levels.

`line_color` A character specifying the color of the line connecting observations.

`line_type` A character specifying the line type for observation line.

`vline_color_onset` A character specifying the color of the vertical seasonal onset line.

`vline_linetype_onset` A character specifying the line type for the seasonal onset line.

`vline_color_offset` A character specifying the color of the vertical seasonal offset line.

`vline_linetype_offset` A character specifying the line type for the seasonal offset line.

`y_scale_labels` A function to format y-axis labels.

`theme_custom` A function with a ggplot2 theme, specifying the theme to apply to the plot.

`k` An integer specifying the window size used to create the `tsd_onset` object.

`skip_current_season` A logical. Do you want to skip your current season?

`breaks_y_axis` A numeric specifying how many breaks to show on the y-axis.

Value

A 'ggplot' object for visualizing the `tsd` data.

A 'ggplot' object for visualizing the `tsd_onset` data.

A 'ggplot' object for visualizing the `tsd_onset_and_burden` data for the current season.

A 'ggplot' object for visualizing the `tsd_growth_warning` data.

Examples

```
set.seed(345)
# Create an example `tsd` object
time_series <- generate_seasonal_data()
autoplot(time_series)

# Create an `tsd_onset` object
time_series_with_onset <- seasonal_onset(
  tsd = time_series,
  k = 3,
  level = 0.95,
  family = "quasipoisson"
)
autoplot(time_series_with_onset)


# Define `disease_threshold`
disease_threshold <- 150

# Create a `tsd_onset_and_burden` object
```

```
tsd_onset_burden <- combined_seasonal_output(
  tsd = time_series,
  disease_threshold = disease_threshold
)
autoplot(tsd_onset_burden)

# Create an `tsd_onset` object
tsd_onset <- seasonal_onset(
  tsd = time_series,
  k = 5,
  family = "quasipoisson",
  season_start = 21,
  only_current_season = FALSE
)

tsd_growth_warning <- consecutive_growth_warnings(tsd_onset)

autoplot(tsd_growth_warning)
```

`combined_seasonal_output` *Compute seasonal onset and burden levels from seasonal time series observations.*

Description

This function performs automated and early detection of seasonal epidemic onsets and estimates the burden levels from time series dataset stratified by season. The seasonal onset estimates growth rates for consecutive time intervals and calculates the average sum of cases/incidence in consecutive time intervals (`k`). The burden levels use the previous seasons to estimate the levels of the current season. Output will be in incidence if `population` and `incidence` are assigned in input.

Usage

```
combined_seasonal_output(
  tsd,
  disease_threshold = 20,
  family = c("quasipoisson", "poisson"),
  family_quant = c("lnorm", "weibull", "exp"),
  season_start = 21,
  season_end = season_start - 1,
  only_current_season = TRUE,
  multiple_waves = FALSE,
  burden_level_decrease = c("low", "very low", "medium", "high"),
  steps_with_decrease = 2,
  ...
)
```

Arguments

tsd	A tsd object containing time series data
disease_threshold	A number specifying the threshold for considering a disease outbreak. Should be given as incidence if population and incidence_denominator are in the tsd object else as cases. For seasonal onset it defines the per time-step disease threshold that has to be surpassed to possibly trigger a seasonal onset alarm. If the average observation count in a window of size k exceeds disease_threshold, a seasonal onset alarm can be triggered. For burden levels it defines the per time-step disease threshold that has to be surpassed for the observation to be included in the level calculations.
family	A character string specifying the family for modeling. Choose between 'poisson', or 'quasipoisson'. Must be one of: character, family-generator, or family object. This is passed to 'seasonal_onset()'.
family_quant	A character string specifying the family for modeling burden levels.
season_start, season_end	Integers giving the start and end weeks of the seasons to stratify the observations by.
only_current_season	Should the output only include results for the current season?
multiple_waves	A logical. Should the output contain multiple waves?
burden_level_decrease	A character string specifying the burden breakpoint the observations should decrease under to reach seasonal_offset or before a new increase in observations can call a new wave onset if multiple_waves are TRUE. Choose between; "very low", "low", "medium", or "high".
steps_with_decrease	An integer specifying in how many time steps (days, weeks, months) the decrease should be observed under the burden_level_decrease (if there is a sudden decrease followed by an increase it could e.g. be due to testing).
...	Arguments passed to seasonal_burden_levels(), fit_percentiles() and seasonal_onset() functions.

Value

An tsd_onset_and_burden object containing two lists:

onset_output:

A tsd_onset object containing:

- 'reference_time': The time point for which the growth rate is estimated.
- 'cases': The cases at reference time point.
- 'population': The population at reference time point.
- 'incidence': The incidence at reference time point.
- 'season': The stratification of observables in corresponding seasons.

- 'growth_rate': The estimated growth rate.
- 'lower_growth_rate': The lower bound of the growth rate's confidence interval.
- 'upper_growth_rate': The upper bound of the growth rate's confidence interval.
- 'growth_warning': Logical. Is the growth rate significantly higher than zero?
- 'average_observation_window': The average of cases or incidence within the time window.
- 'average_observation_warning': Logical. Does the average observations exceed the disease threshold?
- 'seasonal_onset_alarm': Logical. Is there a seasonal onset alarm?
- 'skipped_window': Logical. Was the window skipped due to missing observations?
- 'converged': Logical. Was the IWLS judged to have converged?
- 'seasonal_onset': Logical. The first detected seasonal onset in the season.
- Attributes: `time_interval` and `incidence_denominator`.

As extra the `tsd_onset` object will for each season contain a `seasonal_offset` variable:

- 'seasonal_offset': Logical. The first detected seasonal offset in the season.

If multiple waves is selected the `tsd_onset` object will also contain:

- 'wave_number': The wave number in the time series data.
- 'wave_starts': Logical. Did a new wave start?
- 'wave_ends': Logical. Did the wave end?
- 'decrease_counter': How many consecutive time intervals have decreased below the selected burden breakpoint.
- 'decrease_value': A numeric specifying the selected burden breakpoint value to fall below for ending the wave.

burden_output:

A `tsd_burden_levels` object containing:

- 'season': The season that burden levels are calculated for.
- 'high_conf_level': (only for intensity_level method) The conf_level chosen for the high level.
- 'conf_levels': (only for peak_level method) The conf_levels chosen to fit the 'low', 'medium', 'high' levels.
- 'values': A named vector with values for 'very low', 'low', 'medium', 'high' levels.
- 'optim' A list containing:
  - 'par': The fit parameters for the chosen family.
    - par_1:
      - For 'weibull': Shape parameter.
      - For 'lnorm': Mean of the log-transformed observations.
      - For 'exp': Rate parameter.
    - 'par_2':
      - For 'weibull': Scale parameter.

        - For 'lnorm': Standard deviation of the log-transformed observations.
        - For 'exp': Not applicable (set to NA).
    - 'obj_value': The value of the objective function - (negative log-likelihood), which represent the minimised objective function value from the optimisation. Smaller value equals better optimisation.
    - 'converged': Logical. TRUE if the optimisation converged.
    - 'family': The distribution family used for the optimization.
        * 'weibull': Uses the Weibull distribution for fitting.
        * 'lnorm': Uses the Log-normal distribution for fitting.
        * 'exp': Uses the Exponential distribution for fitting.
- 'disease_threshold': The input disease threshold, which is also the very low level.
- 'incidence_denominator': The observations per incidence-denominator.
- Attributes: `time_interval` and `incidence_denominator`.

#' Attributes in the `tsd_onset_and_burden` object are: `burden_level_decrease`, `steps_with_decrease` and `multiple_waves`.

Examples

```
# Generate random flu season
generate_flu_season <- function(start = 1, end = 1000) {
  random_increasing_obs <- round(sort(runif(24, min = start, max = end)))
  random_decreasing_obs <- round(rev(random_increasing_obs))

  # Generate peak numbers
  add_to_max <- c(50, 100, 200, 100)
  peak <- add_to_max + max(random_increasing_obs)

  # Combine into a single observations sequence
  observations <- c(random_increasing_obs, peak, random_decreasing_obs)

 return(observations)
}

season_1 <- generate_flu_season()
season_2 <- generate_flu_season()

start_date <- as.Date("2022-05-29")
end_date <- as.Date("2024-05-20")

weekly_dates <- seq.Date(from = start_date,
                         to = end_date,
                         by = "week")

tsd_data <- to_time_series(
  cases = c(season_1, season_2),
  time = as.Date(weekly_dates)
)

# Run the main function
```

```
combined_data <- combined_seasonal_output(tsd_data)
# Print seasonal onset results
print(combined_data$onset_output)
# Print burden level results
print(combined_data$burden_output)
```

`consecutive_growth_warnings` *Create a tsd_growth_warning object to count consecutive significant observations*

Description

This function calculates the number of consecutive significant ("growth_warning") observations, grouping them accordingly. The result is stored in an S3 object of class `tsd_growth_warning`.

Uses data from a `tsd_onset` object (output from `seasonal_onset()`).

`seasonal_onset()` has to be run with arguments;

- season_start
- season_end
- only_current_season = FALSE

Usage

```
consecutive_growth_warnings(onset_output)
```

Arguments

`onset_output`	A `tsd_onset` object returned from `seasonal_onset()`.

Value

An object of class `tsd_growth_warning`, containing; A tibble of processed observations, the significant_counter column specifies when a sequence of significant observation starts and ends. The first number is how many subsequent observations will be significant.

Examples

```
# Generate simulated data of seasonal waves
sim_data <- generate_seasonal_data(
  years = 5,
  start_date = as.Date("2022-05-26"),
  trend_rate = 1.002,
  noise_overdispersion = 2,
  relative_epidemic_concentration = 3
)

# Estimate seasonal onset
```

```
tsd_onset <- seasonal_onset(
  tsd = sim_data,
  season_start = 21,
  season_end = 20,
  only_current_season = FALSE
)

# Get consecutive significant observations
consecutive_growth_warnings(tsd_onset)
```

`epi_calendar` *Determine Epidemiological Season*

Description

This function identifies the epidemiological season, (must span new year) to which a given date belongs. The epidemiological season is defined by a start and end week, where weeks are numbered according to the ISO week date system.

Usage

```
epi_calendar(date, start = 21, end = 20)
```

Arguments

`date`	A date object representing the date to check.
`start`	An integer specifying the start week of the epidemiological season.
`end`	An integer specifying the end week of the epidemiological season.

Value

A character vector indicating the season:

- "out_of_season" if the date is outside the specified season,
- If within the season, the function returns a character string indicating the epidemiological season.

Examples

```
# Check if a date is within the epidemiological season
epi_calendar(as.Date("2023-09-15"), start = 21, end = 20)
# Expected output: "2023/2024"

epi_calendar(as.Date("2023-05-30"), start = 40, end = 20)
# Expected output: "out_of_season"

try(epi_calendar(as.Date("2023-01-15"), start = 1, end = 40))
# Expected error: "`start` must be greater than `end`!"
```

```
epi_calendar(as.Date("2023-10-06"), start = 40, end = 11)
# Expected output: "2023/2024"
```

`estimate_disease_threshold` *Estimate the disease specific threshold of your time series data*

Description

This function estimates the disease specific threshold, based on previous seasons. If the disease threshold is estimated between]0:1] it will be set to 1.

Usage

```
estimate_disease_threshold(
  tsd,
  season_start = 21,
  season_end = season_start - 1,
  skip_current_season = TRUE,
  min_significant_time = 3,
  max_gap_time = 1,
  use_prev_seasons_num = 3,
  pick_significant_sequence = c("longest", "earliest"),
  season_importance_decay = 0.8,
  conf_levels = c(0.25, 0.5, 0.75),
  ...
)
```

Arguments

`tsd`	A `tsd` object containing time series data
`season_start, season_end`	Integers giving the start and end weeks of the seasons to stratify the observations by.
`skip_current_season`	A logical. Do you want to skip your current season?
`min_significant_time`	An integer specifying how many time steps that have to be significant to the sequence to be considered in estimation.
`max_gap_time`	A numeric value specifying how many time steps there is allowed to be non-significant between two significant sequences for maybe considering them as the same sequence. Sometimes e.g. vacations or less testing can lead to false decreases.
`use_prev_seasons_num`	An integer specifying how many previous seasons you want to include in estimation.

pick_significant_sequence
: A character string specifying which significant sequence to pick from each season.
  - longest: The longest sequence of size min_significant_time closest to the peak.
  - earliest: The earliest sequence of size min_significant_time of the season.

season_importance_decay
: A numeric value between 0 and 1, that specifies the weight applied to previous seasons. It is used as season_importance_decay^(number of seasons back), whereby the weight for the most recent season will be season_importance_decay^0 = 1. This parameter allows for a decreasing weight assigned to prior seasons, such that the influence of older seasons diminishes exponentially.

conf_levels
: A numeric vector specifying the confidence levels for parameter estimates. The values have to be unique and in ascending order, the first percentile is the disease specific threshold. Specify one or three confidence levels e.g.: c(0.25) c(0.25, 0.5, 0.75).

...
: Arguments passed to the seasonal_onset() or fit_percentiles() function. only_current_season = FALSE and disease_threshold = NA_real_ cannot be changed in seasonal_onset().

Value

An object of class tsd_disease_threshold, containing; ....

Examples

```
# Generate seasonal data
tsd_data <- generate_seasonal_data(
 years = 3,
 start_date = as.Date("2021-01-01"),
 noise_overdispersion = 3
)

# Estimate disease threshold
estimate_disease_threshold(tsd_data)
```

---

fit_growth_rate *Fit a growth rate model to time series cases.*

Description

This function fits a growth rate model to time series cases and provides parameter estimates along with confidence intervals.

Usage

```
fit_growth_rate(
  cases,
  population = NULL,
  level = 0.95,
  family = c("quasipoisson", "poisson")
)
```

Arguments

cases	An integer vector containing the time series cases.
population	An integer vector containing the time series background population.
level	The confidence level for parameter estimates, a numeric value between 0 and 1.
family	A character string specifying the family for modeling. Choose between 'poisson', or 'quasipoisson'. Must be one of: character, family-generator, or family object.

Value

A list containing:

- 'fit': The fitted growth rate model.
- 'estimate': A numeric vector with parameter estimates, including the growth rate and its confidence interval.
- 'level': The confidence level used for estimating parameter confidence intervals.

Examples

```
# Fit a growth rate model to a time series of counts
# (e.g., population growth)
data <- c(100, 120, 150, 180, 220, 270)
fit_growth_rate(
  cases = data,
  level = 0.95,
  family = "poisson"
)
```

---

fit_percentiles *Fits weighted observations to distribution and returns percentiles*

Description

This function estimates the percentiles of weighted time series cases or incidences. The output contains the percentiles from the fitted distribution.

Usage

```
fit_percentiles(
  weighted_observations,
  conf_levels = c(0.5, 0.9, 0.95),
  family = c("lnorm", "weibull", "exp"),
  optim_method = c("Nelder-Mead", "BFGS", "CG", "L-BFGS-B", "SANN", "Brent"),
  lower_optim = -Inf,
  upper_optim = Inf
)
```

Arguments

`weighted_observations`	A tibble containing two columns of length n; `observation`, which contains either cases or incidences, and `weight`, which is the importance assigned to the observation. Higher weights indicate that an observation has more influence on the model outcome, while lower weights reduce its impact.
`conf_levels`	A numeric vector specifying the confidence levels for parameter estimates. The values have to be unique and in ascending order, that is the lowest level is first and highest level is last.
`family`	A character string specifying the family for modeling. Choose between 'poisson', or 'quasipoisson'. Must be one of: character, family-generator, or family object.
`optim_method`	A character string specifying the method to be used in the optimisation. Lookup `?optim::stats` for details about methods. If using the exp family it is recommended to use Brent as it is a one-dimensional optimisation.
`lower_optim`	A numeric value for the optimisation.
`upper_optim`	A numeric value for the optimisation.

Value

A list containing:

- 'conf_levels': The conf_levels chosen to fit the percentiles.
- 'percentiles': The percentile results from the fit.
- 'par': The fit parameters for the chosen family.
  - par_1:
    - For 'weibull': Shape parameter (k).
    - For 'lnorm': Mean of the log-transformed observations.
    - For 'exp': Rate parameter (rate).
  - 'par_2':
    - For 'weibull': Scale parameter (scale).
    - For 'lnorm': Standard deviation of the log-transformed observations.
    - For 'exp': Not applicable (set to NA).

- 'obj_value': The value of the objective function - (negative log-likelihood), which represent the minimized objective function value from the optimisation. Smaller value equals better optimisation.
- 'converged': Logical. TRUE if the optimisation converged.
- 'family': The distribution family used for the optimization.
  - 'weibull': Uses the Weibull distribution for fitting.
  - 'lnorm': Uses the Log-normal distribution for fitting.
  - 'exp': Uses the Exponential distribution for fitting.

Examples

```
# Create three seasons with random observations
obs <- 10
season <- c("2018/2019", "2019/2020", "2020/2021")
season_num_rev <- rev(seq(from = 1, to = length(season)))
observations <- rep(stats::rnorm(10, obs), length(season))

# Add into a tibble with decreasing weight for older seasons
data_input <- tibble::tibble(
  observation = observations,
  weight = 0.8^rep(season_num_rev, each = obs)
)

# Use the model
fit_percentiles(
  weighted_observations = data_input,
  conf_levels = c(0.50, 0.90, 0.95),
  family= "weibull"
)
```

---

generate_seasonal_data

Generate Simulated Data of Seasonal Waves as a `tsd` *object*

Description

This function generates a simulated dataset of seasonal waves with trend and noise. This function assumes 365 days, 52 weeks, and 12 months per year. Leap years are not included in the calculation.

Usage

```
generate_seasonal_data(
  years = 3,
  start_date = as.Date("2021-05-26"),
  amplitude = 100,
  mean = 100,
  phase = 0,
```

```
  trend_rate = NULL,
  noise_overdispersion = NULL,
  relative_epidemic_concentration = 1,
  time_interval = c("weeks", "days", "months"),
  lower_bound = 1e-06
)
```

Arguments

years	An integer specifying the number of years of data to simulate.
start_date	A date representing the start date of the simulated data.
amplitude	A number specifying the amplitude of the seasonal wave. The output will fluctuate within the range [mean - amplitude, mean + amplitude].
mean	A number specifying the mean of the seasonal wave.
phase	A numeric value (in radians) representing the horizontal shift of the sine wave, hence the phase shift of the seasonal wave. The phase must be between zero and 2*pi.
trend_rate	A numeric value specifying the exponential growth/decay rate.
noise_overdispersion	A numeric value specifying the overdispersion of the generated data. 0 means deterministic, 1 is pure poisson and for values > 1 a negative binomial is assumed.
relative_epidemic_concentration	A numeric that transforms the reference sinusoidal season. A value of 1 gives the pure sinusoidal curve, and greater values concentrate the epidemic around the peak.
time_interval	A character vector specifying the time interval. Choose between 'days', 'weeks', or 'months'.
lower_bound	A numeric value that can be used to ensure that intensities are always greater than zero, which is needed when noise_overdispersion is different from zero.

Value

A tsd object with simulated data containing:

- 'time': The time point for the corresponding data.
- 'cases': The number of cases at the time point.

Examples

```
# Generate simulated data of seasonal waves

#With default arguments
default_sim <- generate_seasonal_data()
plot(default_sim)

#With an exponential growth rate trend
```

```
trend_sim <- generate_seasonal_data(trend_rate = 1.001)
plot(trend_sim)

#With noise
noise_sim <- generate_seasonal_data(noise_overdispersion = 2)
plot(noise_sim)

#With distinct parameters, trend and noise
sim_data <- generate_seasonal_data(
  years = 2,
  start_date = as.Date("2022-05-26"),
  amplitude = 2000,
  mean = 3000,
  trend_rate = 1.002,
  noise_overdispersion = 1.1,
  time_interval = c("weeks")
)
plot(sim_data, time_interval = "2 months")
```

`historical_summary` *Summarises estimates like seasonal peak and onset from all available seasons*

Description

This function summarises peak timing and seasonal onset from estimates in a `tsd_onset` object. This can be useful for investigating if the current season falls within estimates from previous seasons or if it is very distinct from previous seasons.

Uses data from a `tsd_onset` object (output from `seasonal_onset()`).

`seasonal_onset()` has to be run with arguments;

- disease_threshold
- season_start
- season_end
- only_current_season = FALSE

Usage

```
historical_summary(onset_output)
```

Arguments

`onset_output`	A `tsd_onset` object returned from `seasonal_onset()`.

Value

An object of class `historical_summary`, containing:

- Usual time to seasonal peak (weeks after onset)
- The week in which the peak usually falls
- Usual peak intensity
- The week in which the onset usually falls
- Usual onset intensity and growth rate estimates If the season does not have an onset, it will not be included in the summary.

Examples

```
# Generate simulated data of seasonal waves
sim_data <- generate_seasonal_data(
  years = 5,
  start_date = as.Date("2022-05-26"),
  trend_rate = 1.002,
  noise_overdispersion = 1.1
)

# Estimate seasonal onset
tsd_onset <- seasonal_onset(
  tsd = sim_data,
  disease_threshold = 20,
  family = "quasipoisson",
  season_start = 21,
  season_end = 20,
  only_current_season = FALSE
)

# Get historical summary
historical_summary(tsd_onset)
```

---

plot.tsd *Create a complete 'ggplot' appropriate to a particular data type*

Description

This function generates a complete 'ggplot' object suitable for visualizing time series data in `tsd`, `tsd_onset`, `tsd_onset_and_burden` or `tsd_growth_warning` objects.

Usage

```
## S3 method for class 'tsd'
plot(x, ...)

## S3 method for class 'tsd_onset'
```

```
plot(x, ...)

## S3 method for class 'tsd_onset_and_burden'
plot(x, ...)

## S3 method for class 'tsd_growth_warning'
plot(x, ...)
```

Arguments

x	An tsd, tsd_onset, tsd_onset_and_burden or tsd_growth_warning object
...	Additional arguments passed to autoplot().

Value

A 'ggplot' object for visualizing output from desired method.

See Also

autoplot()

Examples

```
# set.seed(321)
# Create and plot `tsd` object
tsd_obj <- generate_seasonal_data(
  years = 1,
  phase = 1,
  start_date = as.Date("2021-10-18")
)
plot(tsd_obj)

disease_threshold <- 150

# Create and plot `tsd_onset` object
tsd_onset_obj <- seasonal_onset(
  tsd = tsd_obj,
  k = 3,
  level = 0.95,
  disease_threshold = disease_threshold,
  family = "quasipoisson"
)
plot(tsd_onset_obj)

# Create a `tsd_onset_and_burden` object
tsd_onset_burden_obj <- combined_seasonal_output(
  tsd = tsd_obj,
  disease_threshold = disease_threshold
)
plot(tsd_onset_burden_obj,
     y_lower_bound = ifelse(disease_threshold < 10, 1, 5))
```

```
# Create a `tsd_growth_warning` object
tsd_onset_seasons <- seasonal_onset(
  tsd = tsd_obj,
  season_start = 21,
  family = "quasipoisson",
  only_current_season = FALSE
)
tsd_gr_w <- consecutive_growth_warnings(tsd_onset_seasons)
plot(tsd_gr_w)
```

`predict.tsd_onset` *Predict Cases for Future Time Steps*

Description

This function is used to predict future cases based on a `tsd_onset` object. It uses the `time_interval` attribute from the `tsd_onset` object to make predictions.

Usage

```
## S3 method for class 'tsd_onset'
predict(object, n_step = 3, ...)
```

Arguments

`object`	A `tsd_onset` object created using the `seasonal_onset()` function.
`n_step`	An integer specifying the number of future time steps for which you want to predict cases.
`...`	Additional arguments (not used).

Value

A tibble-like object called `tsd_predict` containing the predicted cases, including reference time, lower confidence interval, and upper confidence interval for the specified number of future time steps.

Examples

```
# Generate predictions of time series data
set.seed(123)
time_series <- generate_seasonal_data(
  years = 1,
  time_interval = "days"
)
# Apply `seasonal_onset` analysis
time_series_with_onset <- seasonal_onset(
  tsd = time_series,
```

```
  k = 7
)
# Predict cases for the next 7 time steps
predict(object = time_series_with_onset, n_step = 7)
```

`seasonal_burden_levels` *Compute burden levels from seasonal time series observations of current season.*

Description

This function estimates the burden levels of time series observations that are stratified by season. It uses the previous seasons to estimate the levels of the current season. The output is results regarding the current season in the time series observations. NOTE: The data must include data for a complete previous season to make predictions for the current season. Observations will be incidence if `population` and `incidence` are available in the `tsd` object.

Usage

```
seasonal_burden_levels(
  tsd,
  family = c("lnorm", "weibull", "exp"),
  season_start = 21,
  season_end = season_start - 1,
  method = c("intensity_levels", "peak_levels"),
  conf_levels = 0.95,
  decay_factor = 0.8,
  disease_threshold = 20,
  n_peak = 6,
  only_current_season = TRUE,
  ...
)
```

Arguments

`tsd`	A `tsd` object containing time series data
`family`	A character string specifying the family for modeling. Choose between 'poisson', or 'quasipoisson'. Must be one of: character, family-generator, or family object.
`season_start, season_end`	Integers giving the start and end weeks of the seasons to stratify the observations by.
`method`	A character string specifying the model to be used in the level calculations. Both model predict the levels of the current series of observations.

- `intensity_levels`: models the risk compared to what has been observed in previous seasons.

- peak_levels: models the risk compared to what has been observed in the n_peak observations each season.

conf_levels
: A numeric vector specifying the confidence levels for parameter estimates. The values have to be unique and in ascending order, (i.e. the lowest level is first and highest level is last). The conf_levels are specific for each method:
  - for intensity_levels only specify the highest confidence level e.g.: 0.95, which is the highest intensity that has been observed in previous seasons.
  - for peak_levels specify three confidence levels e.g.: c(0.4, 0.9, 0.975), which are the three confidence levels low, medium and high that reflect the peak severity relative to those observed in previous seasons.

decay_factor
: A numeric value between 0 and 1, that specifies the weight applied to previous seasons in level calculations. It is used as decay_factor^(number of seasons back), whereby the weight for the most recent season will be decay_factor^0 = 1. This parameter allows for a decreasing weight assigned to prior seasons, such that the influence of older seasons diminishes exponentially.

disease_threshold
: A number specifying the threshold for considering a disease outbreak. Should be given as incidence if population and incidence_denominator are in the tsd object else as cases. It defines the per time-step disease threshold that has to be surpassed for the observation to be included in the level calculations.

n_peak
: A numeric value specifying the number of peak observations to be selected from each season in the level calculations. The n_peak observations have to surpass the disease_threshold to be included.

only_current_season
: Should the output only include results for the current season?

...
: Arguments passed to the fit_percentiles() function.

Value

A tsd_burden_levels object containing:

- 'season': The season that burden levels are calculated for.
- 'high_conf_level': (only for intensity_level method) The conf_level chosen for the high level.
- 'conf_levels': (only for peak_level method) The conf_levels chosen to fit the 'low', 'medium', 'high' levels.
- 'values': A named vector with values for 'very low', 'low', 'medium', 'high' levels.
- 'optim' A list containing:
  - 'par': The fit parameters for the chosen family.
    - par_1:
      - For 'weibull': Shape parameter.
      - For 'lnorm': Mean of the log-transformed observations.
      - For 'exp': Rate parameter.
    - 'par_2':
      - For 'weibull': Scale parameter.

        - For 'lnorm': Standard deviation of the log-transformed observations.
        - For 'exp': Not applicable (set to NA).
    - 'obj_value': The value of the objective function - (negative log-likelihood), which represent the minimised objective function value from the optimisation. Smaller value equals better optimisation.
    - 'converged': Logical. TRUE if the optimisation converged.
    - 'family': The distribution family used for the optimization.
        - 'weibull': Uses the Weibull distribution for fitting.
        - 'lnorm': Uses the Log-normal distribution for fitting.
        - 'exp': Uses the Exponential distribution for fitting.
- 'disease_threshold': The input disease threshold, which is also the very low level.
- 'incidence_denominator': The observations per incidence-denominator.
- Attributes: `time_interval` and `incidence_denominator`.

Examples

```
# Generate random flu season
generate_flu_season <- function(start = 1, end = 1000) {
  random_increasing_obs <- round(sort(runif(24, min = start, max = end)))
  random_decreasing_obs <- round(rev(random_increasing_obs))

  # Generate peak numbers
  add_to_max <- c(50, 100, 200, 100)
  peak <- add_to_max + max(random_increasing_obs)

  # Combine into a single observations sequence
  observations <- c(random_increasing_obs, peak, random_decreasing_obs)

 return(observations)
}

season_1 <- generate_flu_season()
season_2 <- generate_flu_season()

start_date <- as.Date("2022-05-29")
end_date <- as.Date("2024-05-20")

weekly_dates <- seq.Date(from = start_date,
                         to = end_date,
                         by = "week")

tsd_data <- to_time_series(
  cases = c(season_1, season_2),
  time = as.Date(weekly_dates)
)

# Print seasonal burden results
seasonal_burden_levels(tsd_data, family = "lnorm")
```

seasonal_onset *Automated and Early Detection of Seasonal Epidemic Onset*

Description

This function performs automated and early detection of seasonal epidemic onsets on a tsd object. It estimates growth rates and calculates the average sum of cases in consecutive time intervals (k). If the time series data includes population it will be used as offset to adjust the growth rate in the glm, additionally the output will include incidence, population and average sum of incidence.

Usage

```
seasonal_onset(
  tsd,
  k = 5,
  level = 0.95,
  disease_threshold = NA_real_,
  family = c("quasipoisson", "poisson"),
  na_fraction_allowed = 0.4,
  season_start = NULL,
  season_end = season_start - 1,
  only_current_season = NULL
)
```

Arguments

tsd	A tsd object containing time series data
k	An integer specifying the window size for modeling growth rates and average sum of cases.
level	The confidence level for onset parameter estimates, a numeric value between 0 and 1.
disease_threshold	A number specifying the threshold for considering a disease outbreak. Should be given as incidence if population and incidence_denominator are in the tsd object else as cases. It defines the per time-step disease threshold that has to be surpassed to possibly trigger a seasonal onset alarm. If the average observation count in a window of size k exceeds disease_threshold, a seasonal onset alarm can be triggered.
family	A character string specifying the family for modeling. Choose between 'poisson', or 'quasipoisson'. Must be one of: character, family-generator, or family object.
na_fraction_allowed	Numeric value between 0 and 1 specifying the fraction of observations in the window of size k that are allowed to be NA or zero, i.e. without cases, in onset calculations.

`season_start`, `season_end`
: Integers giving the start and end weeks of the seasons to stratify the observations by. If set to `NULL`, it means no stratification by season.

`only_current_season`
: Should the output only include results for the current season?

Value

A `tsd_onset` object containing:

- 'reference_time': The time point for which the growth rate is estimated.
- 'cases': The cases at reference time point.
- 'population': The population at reference time point.
- 'incidence': The incidence at reference time point.
- 'season': The stratification of observables in corresponding seasons.
- 'growth_rate': The estimated growth rate.
- 'lower_growth_rate': The lower bound of the growth rate's confidence interval.
- 'upper_growth_rate': The upper bound of the growth rate's confidence interval.
- 'growth_warning': Logical. Is the growth rate significantly higher than zero?
- 'average_observation_window': The average of cases or incidence within the time window.
- 'average_observation_warning': Logical. Does the average observations exceed the disease threshold?
- 'seasonal_onset_alarm': Logical. Is there a seasonal onset alarm?
- 'skipped_window': Logical. Was the window skipped due to missing observations?
- 'converged': Logical. Was the IWLS judged to have converged?
- 'seasonal_onset': Logical. The first detected seasonal onset in the season.
- Attributes: `time_interval` and `incidence_denominator`.

Examples

```
# Create a tibble object from sample data
tsd_data <- to_time_series(
  cases = c(100, 120, 150, 180, 220, 270),
  time = seq(from = as.Date("2023-01-01"), by = "1 week", length.out = 6)
)

# Estimate seasonal onset with a 3-day window
seasonal_onset(
  tsd = tsd_data,
  k = 3,
  level = 0.975,
  disease_threshold = 5,
  na_fraction_allowed = 0.4,
  season_start = 21,
  season_end = 20,
  only_current_season = FALSE
)
```

summary.tsd_burden_levels *Summary method for* tsd_burden_levels *objects*

Description

Summarize key results from a seasonal burden levels analysis.

Usage

```
## S3 method for class 'tsd_burden_levels'
summary(object, ...)
```

Arguments

object	An object of class 'tsd_burden_levels' containing the results of a seasonal_burden_levels analysis.
...	Additional arguments (not used).

Value

This function is used for its side effect, which is printing the burden levels.

Examples

```
# Create a `tsd` object
tsd_data <- generate_seasonal_data()

# Create a `tsd_burden_levels` object
tsd_burden_levels <- seasonal_burden_levels(
  tsd = tsd_data
)
# Print the summary
summary(tsd_burden_levels)
```

summary.tsd_onset *Summary method for* tsd_onset *objects*

Description

Summarize key results from a seasonal onset analysis.

Usage

```
## S3 method for class 'tsd_onset'
summary(object, ...)
```

Arguments

object	An object of class 'tsd_onset' containing the results of a seasonal_onset analysis.
...	Additional arguments (not used).

Value

This function is used for its side effect, which is printing a summary message to the console.

Examples

```
# Create a `tsd` object
tsd_data <- generate_seasonal_data()

# Create a `tsd_onset` object
tsd_onset <- seasonal_onset(
  tsd = tsd_data,
  k = 3,
  disease_threshold = 100,
  season_start = 21,
  season_end = 20,
  level = 0.95,
  only_current_season = TRUE
)
# Print the summary
summary(tsd_onset)
```

---

to_time_series *Create a tibble-like* tsd *(time-series data) object from time series data and corresponding dates.*

Description

This function takes cases and the corresponding date vector (time) and converts it into a tsd object, which is a time series data structure that can be used for time series analysis. If incidence is added, it will be used as observation in all future use of the aedseo package on the defined tsd object.

Options:

- incidence can be calculated if also supplying cases, population, and incidence_denominator.
- cases can be calculated if also supplying incidence, population and incidence_denominator.
- If background population changes during the time series, it is used to adjust the growth rate in seasonal_onset().

Usage

```
to_time_series(
  cases = NULL,
  incidence = NULL,
  population = NULL,
  incidence_denominator = if (is.null(population)) NA_real_ else 1e+05,
  time,
  time_interval = c("weeks", "days", "months")
)
```

Arguments

cases	An integer vector containing the time series cases.
incidence	A numeric vector containing the time series incidences. With the given incidence_denominator.
population	An integer vector containing the time series background population.
incidence_denominator	An integer >= 1, specifying the observations per incidence-denominator.
time	A date vector containing the corresponding dates.
time_interval	A character vector specifying the time interval. Choose between 'days', 'weeks', or 'months'.

Value

A `tsd` object containing:

- 'time': The time point for the corresponding data.
- 'cases': The number of cases at the time point.
- 'incidence': The incidence per `incidence_denominator` at the time point. (optional)
- 'population': The background population for the cases at the time point. (optional)

Examples

```
# Create a `tsd` object with only cases
tsd_cases <- to_time_series(
  cases = c(10, 15, 20, 18),
  time = seq(from = as.Date("2023-01-01"), by = "1 week", length.out = 4)
)

# Create a `tsd` object with incidence from cases, population and default incidence_denominator
tsd_calculate_incidence <- to_time_series(
  cases = c(100, 120, 130, 150),
  time = seq(from = as.Date("2023-01-01"), by = "1 week", length.out = 4),
  population = c(3000000, 3000000, 3000000, 3000000)
)

# Create a `tsd` object with cases from incidence, population and default incidence_denominator
tsd_calculate_cases <- to_time_series(
```

```
  incidence = c(5, 7.8, 8, 8.5),
  time = seq(from = as.Date("2023-01-01"), by = "1 week", length.out = 4),
  population = c(3000000, 3000000, 3000000, 3000000)
)
```

Index