Package 'dendRoAnalyst'

May 20, 2026

Type Package

Date 2026-05-18

Title A Tool for Processing and Analyzing Dendrometer Data

Version 0.1.6

Maintainer Sugam Aryal <sugam.aryal@fau.de>

Description There are various functions for managing and cleaning data before the application of different approaches. This includes identifying and erasing sudden jumps in dendrometer data not related to environmental change, identifying the time gaps of recordings, and changing the temporal resolution of data to different frequencies. Furthermore, the package calculates daily statistics of dendrometer data, including the daily amplitude of tree growth. Various approaches can be applied to separate radial growth from daily cyclic shrinkage and expansion due to uptake and loss of stem water. In addition, it identifies periods of consecutive days with user-defined climatic conditions in daily meteorological data, then check what trees are doing during that period.

License GPL-3

Encoding UTF-8

LazyData true

Depends R (>= 4.1.0), base

Imports stats, tidyverse, dplyr, ggplot2, lubridate, readxl, tibble, tidyr, zoo, forecast, mgcv, minpack.lm, pspline, moments, signal, readr, boot, rlang, changepoint, WaveletComp

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

NeedsCompilation no

Author Sugam Aryal [aut, cre, dtc],
Martin Häusser [aut],
Jussi Grießinger [aut],
Ze-Xin Fan [aut],
Achim Bräuning [aut, dgs]

Repository CRAN

Date/Publication 2026-05-20 14:40:02 UTC

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

RoxygenNote 7.3.3

Contents

`clim.twd` *Calculate relative dendrometer change during adverse climate periods and the following normal phase*

Description

Identifies adverse climate periods from a climate time series and calculates relative dendrometer change from the start of each adverse phase.

In addition to the adverse phase itself, the function also defines a *complete period* for each event ID:

- the adverse climate phase, and
- the following normal climate phase until the next adverse phase starts (or until the end of the common series).

Relative dendrometer change is always calculated from the first day of the adverse phase for that ID.

The function returns:

- `adverse_change`: relative change only during adverse phases,
- `normal_change`: relative change only during the following normal phases,
- `full_period_change`: relative change across adverse + following normal phase,
- `continuous_full_period_change`: same as `full_period_change`, but values do not reset to 0 at the next adverse phase and instead continue from the end of the previous full period,
- `period_info`: per-ID and per-tree summary metrics.

Usage

```
clim.twd(
  df,
  Clim,
  dailyValue = "max",
  thresholdClim = "<10",
  thresholdDays = ">5",
  showPlot = TRUE
)
```

Arguments

df	A data frame with the first column containing date-time stamps (yyyy-mm-dd HH:MM:SS, convertible to POSIXct) and subsequent columns containing dendrometer series.
Clim	A data frame with the first column containing dates and the second column containing the climate variable of interest.
dailyValue	Character. Daily aggregation statistic used when resampling dendrometer data. One of "max", "min", "mean", or "sum". Default is "max".
thresholdClim	Character string defining the climate threshold. Supported forms are: • "<5", "<=5", ">5", ">=5", "==5" • "between(5,10)" • "5:10"
thresholdDays	Character string defining the minimum duration threshold for consecutive adverse days. Must use a single-threshold form such as ">5" or ">=3".
showPlot	Logical. If TRUE, diagnostic plots are produced and the user can interactively choose IDs and phase type ("adverse", "normal", or "both"). Default is TRUE.

Details

The algorithm works as follows:

1. Climate data are thresholded to identify adverse days.
2. Consecutive adverse days are grouped into candidate adverse runs.
3. Only runs satisfying thresholdDays are retained as true adverse events.
4. For each retained adverse event, the following normal period is defined as all non-adverse days until the next adverse event starts (or the end of the series).
5. Dendrometer data are resampled to daily resolution using dendro.resample.
6. Relative dendrometer change is calculated from the first day of the adverse phase for each ID.

Threshold syntax:

- single-threshold forms such as "<5", "<=10", ">3", ">=0", "==0",
- range forms for thresholdClim such as "between(5,10)" or "5:10", both interpreted inclusively.

thresholdDays must use a single-threshold form such as ">5".

Value

A list of class "clim_twd_output" containing:

adverse_change A data frame of relative dendrometer change during adverse phases only.

normal_change A data frame of relative dendrometer change during the following normal phases only.

full_period_change A data frame of relative dendrometer change across adverse + following normal phase, reset to 0 at each adverse start.

continuous_full_period_change A data frame similar to `full_period_change`, but values continue from the end of the previous full period instead of resetting to 0.

period_info A data frame with one row per ID and tree, containing adverse and normal phase boundaries, the date and magnitude of maximum adverse decline during the adverse phase, lags to first drop below zero, lags to maximum adverse decline, and lags to return to zero.

phase_table A data frame with adverse and normal phase boundaries for each ID.

Note

If the dendrometer and climate datasets have different time spans, analysis is automatically restricted to their common overlap.

References

Raffelsbauer V, Spannl S, Peña K, Pucha-Cofrep D, Steppe K, Bräuning A (2019). Tree Circumference Changes and Species-Specific Growth Recovery After Extreme Dry Events in a Montane Rainforest in Southern Ecuador. *Frontiers in Plant Science*, 10, 342. doi:10.3389/fpls.2019.00342

See Also

`dendro.resample`, `phase.zg`, `phase.sc`

Examples

```
data(gf_nepa17)
data(ktm_rain17)

rel_out <- clim.twd(
  df = gf_nepa17,
  Clim = ktm_rain17,
  dailyValue = "max",
  thresholdClim = "<10",
  thresholdDays = ">5",
  showPlot = FALSE
)

head(rel_out$adverse_change)
head(rel_out$normal_change)
head(rel_out$full_period_change)
head(rel_out$continuous_full_period_change)
head(rel_out$period_info)

rel_out2 <- clim.twd(
  df = gf_nepa17,
  Clim = ktm_rain17,
  thresholdClim = "between(5,10)",
  thresholdDays = ">3",
  showPlot = FALSE
)
```

`clim.twd.stats` *Calculate tree-, species-, or site-level statistics from clim.twd output*

Description

Calculates grouped statistics from the output of `clim.twd()`.

The function can summarize relative dendrometer change trajectories and event-level metrics:

- by tree,
- by species,
- by site,
- or by species within site.

It can also restrict IDs to specific months, years, or custom day-of-year windows based on the adverse-phase start date of each ID.

Usage

```
clim.twd.stats(
  x,
  tree_info = NULL,
  response = c("adverse_change", "normal_change", "full_period_change",
    "continuous_full_period_change"),
  group_by = c("tree", "species", "site", "species_site"),
  center = c("mean", "median"),
  conf_level = 0.95,
  months = NULL,
  years = NULL,
  doy_window = NULL,
  year_window = NULL,
  include_tree_series = TRUE
)
```

Arguments

`x`	An object of class `"clim_twd_output"` returned by `clim.twd()`.
`tree_info`	Optional metadata table describing trees. It must contain a column named `tree`, and may additionally contain `species` and `site`. A named character vector is also accepted for species-only mapping, where names are trees and values are species.
`response`	Character. Which response table from `clim.twd()` should be summarized: **‘"adverse_change"‘** Only the adverse phase. **‘"normal_change"‘** Only the following normal phase. **‘"full_period_change"‘** Adverse + following normal phase, reset to 0 at each adverse start.

‘"continuous_full_period_change"‘ Adverse + following normal phase, continuing from the previous full period instead of resetting to 0.

`group_by` Character. Grouping level for trajectory summaries. One of `"tree"`, `"species"`, `"site"`, or `"species_site"`.

`center` Character. Central tendency used for grouped trajectories and grouped period metrics. One of `"mean"` or `"median"`.

`conf_level` Numeric confidence/limit level. Default is `0.95`. Empirical limits are returned as the lower and upper quantiles corresponding to this level.

`months` Optional numeric vector of months (1–12). Only IDs whose adverse-phase start falls in these months are retained.

`years` Optional numeric vector of years. Only IDs whose adverse-phase start year falls in these years are retained.

`doy_window` Optional numeric vector of length 2 defining the allowed day-of-year window for adverse-phase start. Wrapped windows are supported, for example `c(300, 50)`.

`year_window` Optional numeric vector of length 2 defining the allowed year range for adverse-phase start.

`include_tree_series`
Logical. If `TRUE`, the filtered long-format tree-level data are stored in the output.

Details

Temporal filtering is based on the `adverse_start` date in `x$phase_table`. Multiple filters are combined, so for example users can simultaneously restrict to:

- specific months,
- specific years,
- a day-of-year window,
- and a year range.

If `group_by = "species"`, `"site"`, or `"species_site"`, `tree_info` must provide the required columns.

Value

A list of class `"clim_twd_stats"` with:

trajectory_summary Grouped time-series summary for each ID and date, including central tendency, SD bands, and empirical 95% limits.

id_summary Grouped per-ID summary derived from `x$period_info`.

selected_ids IDs retained after temporal filtering.

phase_table Filtered phase table.

tree_info Metadata table used for grouping.

long_data Filtered long-format tree-level response table, if `include_tree_series = TRUE`.

settings List of settings used to generate the summaries.

Examples

```
rel_out <- clim.twd(
  df = gf_nepa17,
  Clim = ktm_rain17,
  thresholdClim = "<10",
  thresholdDays = ">5",
  showPlot = FALSE
)

# tree-level statistics
st1 <- clim.twd.stats(
  rel_out,
  response = "full_period_change",
  group_by = "tree"
)

summary(st1)
plot(st1, type = "trajectory")

# species metadata
tree_info <- data.frame(
  tree = c("T2", "T3"),
  species = c("Pinus", "Pinus"),
  site = c("Ktm", "Ktm")
)

# species-level summaries restricted to IDs starting in months 6 to 8
st2 <- clim.twd.stats(
  rel_out,
  tree_info = tree_info,
  response = "full_period_change",
  group_by = "species",
  center = "median",
  months = 6:8
)

summary(st2)
plot(st2, type = "trajectory", band = "limit95")

# species within site
st3 <- clim.twd.stats(
  rel_out,
  tree_info = tree_info,
  response = "continuous_full_period_change",
  group_by = "species_site",
  doy_window = c(150, 250),
  year_window = c(2017, 2018)
)
```

clim.twd.test *Statistical testing for clim.twd output*

Description

Performs hypothesis tests on parameters derived from clim.twd() output.

This function is designed for scenarios such as:

- species comparison as a whole,
- species comparison by year,
- site comparison as a whole,
- site comparison by year,
- species-within-site comparison,
- ID-wise comparisons.

The function uses the period_info table from clim.twd() and tests selected parameters across groups such as trees, species, sites, or species-site combinations.

Usage

```
clim.twd.test(
  x,
  tree_info = NULL,
 parameter = c("lag_to_below_zero", "lag_to_max_adverse_decline", "lag_to_return_zero",
   "max_adverse_decline_value", "change_end_adverse", "change_end_full_period",
    "continuous_end_full_period"),
  compare_by = c("tree", "species", "site", "species_site"),
  within = c("all", "year", "ID"),
  test_method = c("auto", "t_test", "welch_t", "wilcox", "anova", "kruskal"),
  ids = NULL,
  years = NULL,
  months = NULL,
  doy_window = NULL,
  year_window = NULL,
  pairwise = TRUE,
  p_adjust_method = "BH",
  min_n_per_group = 2
)
```

Arguments

x	An object of class "clim_twd_output" returned by clim.twd().
tree_info	Optional metadata table describing trees. It must contain a column named tree, and may additionally contain species and site. A named character vector is also accepted for species-only mapping, where names are trees and values are species.

`parameter` Character. Parameter from `x$period_info` to test. Supported examples include:

‘"lag_to_below_zero"‘ Lag from adverse start until relative change first drops below zero.

‘"lag_to_max_adverse_decline"‘ Lag from adverse start until maximum adverse decline is reached.

‘"lag_to_return_zero"‘ Lag from adverse start until relative change returns to zero or above.

‘"max_adverse_decline_value"‘ Most negative relative change during the adverse phase.

‘"change_end_adverse"‘ Relative change at the end of the adverse phase.

‘"change_end_full_period"‘ Relative change at the end of the full period.

‘"continuous_end_full_period"‘ Continuous end value across periods.

`compare_by` Character. Grouping variable to compare. One of `"tree"`, `"species"`, `"site"`, or `"species_site"`.

`within` Character. Defines the comparison scenario:

‘"all"‘ Compare groups across all retained IDs.

‘"year"‘ Compare groups separately within each adverse-start year.

‘"ID"‘ Compare groups separately within each ID.

`test_method` Character. One of `"auto"`, `"t_test"`, `"welch_t"`, `"wilcox"`, `"anova"`, or `"kruskal"`. With `"auto"`, the function uses:

- Welch t-test or one-way ANOVA when group distributions look approximately normal,
- Wilcoxon or Kruskal-Wallis otherwise.

`ids` Optional numeric vector of IDs to retain before testing.

`years` Optional numeric vector of years to retain based on `adverse_start`.

`months` Optional numeric vector of months (1–12) to retain based on `adverse_start`.

`doy_window` Optional numeric vector of length 2 defining the allowed day-of-year window of `adverse_start`. Wrapped windows such as `c(300, 50)` are supported.

`year_window` Optional numeric vector of length 2 defining the allowed year range for `adverse_start`.

`pairwise` Logical. If `TRUE`, pairwise post-hoc tests are computed for strata with more than two groups when applicable.

`p_adjust_method`
Character. P-value adjustment method for pairwise tests. Default is `"BH"`.

`min_n_per_group`
Minimum number of non-missing observations required per group before testing. Default is `2`.

Value

A list of class `"clim_twd_test"` containing:

data_used Filtered test data used for the analyses.

test_results Main test results by stratum.

pairwise_results Pairwise comparison table where available.

group_summary Descriptive summaries by stratum and group.

settings Settings used for the analysis.

Examples

```
rel_out <- clim.twd(
  df = gf_nepa17,
  Clim = ktm_rain17,
  thresholdClim = "<10",
  thresholdDays = ">5",
  showPlot = FALSE
)

tree_info <- data.frame(
  tree = c("T2", "T3"),
  species = c("Pinus", "Pinus"),
  site = c("Ktm", "Ktm")
)

# species comparison across all retained IDs
tst1 <- clim.twd.test(
  rel_out,
  tree_info = tree_info,
  parameter = "max_adverse_decline_value",
  compare_by = "species",
  within = "all"
)

summary(tst1)
plot(tst1)

# species comparison by year
tst2 <- clim.twd.test(
  rel_out,
  tree_info = tree_info,
  parameter = "lag_to_return_zero",
  compare_by = "species",
  within = "year"
)

# site comparison by year
tst3 <- clim.twd.test(
  rel_out,
  tree_info = tree_info,
  parameter = "change_end_full_period",
  compare_by = "site",
  within = "year",
  test_method = "kruskal"
)
```

`daily.data` *Daily statistics of dendrometer data*

Description

Computes daily statistics from high-frequency dendrometer time series. For each day, it extracts the minimum and maximum values, their times of occurrence, daily mean and median, daily amplitude, the signed lag between the time of maximum and minimum, the day-to-day change in the daily maximum, and a daily status indicating whether the day is growing, shrinking, or stable relative to the previous day.

Usage

```
daily.data(df, TreeNum)
```

Arguments

`df`	A data frame with the first column containing date-time stamps convertible to `POSIXct` and subsequent columns containing dendrometer measurements.
`TreeNum`	Integer. The index of the tree (column) to analyze. `TreeNum = 1` refers to the first dendrometer column after the time column.

Details

The function requires a data frame with a time column in the first column and one or more dendrometer series in the following columns. The user selects the series using `TreeNum`.

The returned object has class `"daily_output"`, so it can be plotted directly with `plot()`.

The column `Max_diff` is computed as:

$$Max\_diff_t = Max_t - Max_{t-1}$$

The column `Day_status` is derived from `Max_diff`:

- `"growing"` if `Max_diff > 0`
- `"shrinking"` if `Max_diff < 0`
- `"stable"` if `Max_diff = 0`

The first day has `NA` for `Max_diff` and `Day_status`.

Value

A tibble of class `"daily_output"` containing:

DATE Calendar date.

Min Daily minimum value.

Time_min Time of day of minimum value.

Max Daily maximum value.

Time_max Time of day of maximum value.

mean Daily mean value.

median Daily median value.

amplitude Daily amplitude = Max - Min.

Time_min_h Time of minimum expressed in decimal hours.

Time_max_h Time of maximum expressed in decimal hours.

lag_h Signed difference in hours: Time_max_h - Time_min_h.

Remarks `"*"` if `Time_max` > `Time_min`, otherwise `""`.

Max_diff Difference between today's maximum and the previous day's maximum.

Day_status `"growing"`, `"shrinking"`, `"stable"`, or `NA`.

Note

The object returned by `daily.data()` can be plotted using `plot()` because it is assigned class `"daily_output"`.

References

King G, Fonti P, Nievergelt D, Büntgen U, Frank D (2013) Climatic drivers of hourly to yearly tree radius variations along a 6°C natural warming gradient. *Agricultural and Forest Meteorology* 168:36–46. doi:10.1016/j.agrformet.2012.08.002

Examples

```
data(nepa17)
daily_stats <- daily.data(df = nepa17[1:1000, ], TreeNum = 1)
head(daily_stats, 10)
```

`dendro.resample` *Resampling temporal resolution of dendrometer and climate data*

Description

This function changes the temporal resolution of dendrometer and climate time series. It supports both aggregation to coarser resolutions and interpolation to finer resolutions. The target resolution can be defined in minutes, hours, days, weeks, or month-end frequency. Aggregation can be performed using `mean`, `max`, `min`, `sum`, or `count` (number of non-missing values). When a finer temporal resolution than the original data is requested, the function can interpolate the series to the new time step using linear interpolation.

Usage

```
dendro.resample(
  df,
  by,
  value = "mean",
  method = c("auto", "aggregate", "interpolate")
)
```

Arguments

`df`	A dataframe with the first column containing date-time values in the format `yyyy-mm-dd HH:MM:SS`.
`by`	A character string defining the target temporal resolution. Supported values are: • `"M"` for minutes • `"H"` for hours • `"D"` for days • `"W"` for weeks • `"ME"` for month-end • values such as `"10M"`, `"30M"`, `"5H"`, `"2D"`, or `"3W"` for custom multiples
`value`	A character string defining the aggregation function. Supported values are `"mean"`, `"max"`, `"min"`, `"sum"`, and `"count"`. The `"count"` option returns the number of non-NA values within each aggregation interval. It is only valid when `method = "aggregate"`.
`method`	A character string defining how resampling should be performed. Options are: • `"auto"`: automatically aggregates when the requested resolution is coarser than the input data, and interpolates when the requested resolution is finer • `"aggregate"`: always aggregates values within each target interval • `"interpolate"`: always interpolates to the requested regular time grid

Value

A dataframe with resampled data. The first column contains the new time stamps and the remaining columns contain the resampled variables.

Examples

```
library(dendRoAnalyst)
data(nepa17)

# Monthly resampling using maximum values
resample_ME <- dendro.resample(df = gf_nepa17, by = "ME", value = "max")
head(resample_ME, 10)

# Daily resampling using mean values
resample_D <- dendro.resample(df = gf_nepa17, by = "D", value = "mean")

# Count non-missing values per day
resample_count <- dendro.resample(df = gf_nepa17, by = "D", value = "count",
                                  method = "aggregate")

# Interpolate to 30-minute resolution
resample_30M <- dendro.resample(df = gf_nepa17, by = "30M", value = "mean",
                                method = "auto")
```

`dendro.truncate` *Truncation of dendrometer data*

Description

Truncates dendrometer data to a user-defined calendar-year and day-of-year window.

The first column of `df` is treated as the date-time column and is renamed internally to `TIME`. The date-time column may be a `Date`, `POSIXct`, or character vector in the format `"yyyy-mm-dd HH:MM:SS"`. Character dates in the format `"yyyy-mm-dd"` are also accepted.

This version supports leap years correctly. For example, `DOY = 366` is valid for leap years such as 2016, 2020, or 2024, but invalid for non-leap years.

Usage

```
dendro.truncate(df, CalYear, DOY)
```

Arguments

`df`	A data frame where the first column contains date or date-time values. The first column is renamed to `TIME` in the returned data.
`CalYear`	Numeric value or numeric vector of length two giving the calendar year or start and end calendar years.
`DOY`	Numeric value or numeric vector of length two giving the day of year or start and end day of year.

Details

The truncation rules are:

- If `CalYear` has length one and `DOY` has length one, data from that single day are returned.
- If `CalYear` has length one and `DOY` has length two, data from the first DOY to the second DOY within the same year are returned.
- If `CalYear` has length two and `DOY` has length one, data from that DOY in the first year to the same DOY in the second year are returned.
- If `CalYear` has length two and `DOY` has length two, data from the first DOY of the first year to the second DOY of the second year are returned.

The function uses real calendar dates internally rather than row positions. Therefore, leap-year DOY 366 is handled correctly.

Value

A tibble containing the truncated dendrometer data.

Examples

```
library(dendRoAnalyst)
data(nepa)

# Extract data from DOY 20 to 50 in 2017
trunc1 <- dendro.truncate(df = nepa, CalYear = 2017, DOY = c(20, 50))
head(trunc1, 10)

# Leap-year example, if the data contain year 2020
# trunc2 <- dendro.truncate(df = df2020, CalYear = 2020, DOY = c(1, 366))
```

`dm.detrend.fit` *Detrend and standardize dendrometer series from growth-fit residuals*

Description

Creates detrended standardized dendrometer series from objects returned by [dm.growth.fit()] or [dm.growth.fit.double()].

The function compares the original daily dendrometer series (‘x$original_daily_data‘) with the fitted growth curve from ‘x$fitted_data‘. Because fitted values are stored on the processed scale used for modelling, the function first reconstructs fitted values on the original daily scale by adding back the seasonal baseline (the first non-missing original daily value of each series within each vegetation season).

Residuals are then calculated as:

$$residual = observed_{daily} - fitted_{original\ scale}$$

To obtain a detrended standardized series with mean equal to 1 and no negative values, residuals are transformed within each `series` × `season_label` as:

$$z = \frac{residual - \min(residual) + \epsilon}{\text{mean}(residual - \min(residual) + \epsilon)}$$

where $\epsilon > 0$ is a small constant ensuring strictly positive values. If all residuals within a season are identical, the standardized series becomes a constant vector of 1.

Usage

```
dm.detrend.fit(
  x,
  series = NULL,
  seasons = NULL,
  epsilon = 1e-06,
  keep_na_rows = FALSE
)
```

Arguments

`x`	An object of class `"dm_growth_fit"` returned by [dm.growth.fit()] or [dm.growth.fit.double()].
`series`	Optional character vector of dendrometer series to retain. Default is `NULL`, meaning all available series are used.
`seasons`	Optional character vector of vegetation-season labels to retain. Default is `NULL`, meaning all available seasons are used.
`epsilon`	Small positive constant added after shifting residuals by their seasonal minimum. Default is `1e-6`. Use `0` to allow exact zeros.
`keep_na_rows`	Logical. If `TRUE`, rows are retained even when the observed or fitted value is missing; detrended values remain `NA`. If `FALSE` (default), rows with missing observed or fitted values are removed from the long output before standardization.

Details

The function uses `x$original_daily_data`, which must be present in the input object. If the object was created by an older version of [dm.growth.fit()] or [dm.growth.fit.double()] that did not store `original_daily_data`, the function will stop with an informative error.

Standardization is carried out separately within each vegetation season. This means the mean of the detrended standardized series is 1 for each `series` × `season_label`, not necessarily across the entire dataset.

Value

A list of class `"dm_detrended"` with elements:

call The matched function call.

detrended_data Wide-format tibble with metadata columns and one detrended standardized series column per dendrometer.

detrended_long Long-format tibble containing original daily values, reconstructed fitted values on the original scale, residuals, shifted residuals, and detrended standardized values.

parameters Per-series and per-season summary table containing baseline values and residual standardization parameters.

source_fit_statistics The original `x$fit_statistics` table.

Examples

```
data(gf_nepa17)

fit1 <- dm.growth.fit(
  df = gf_nepa17,
  TreeNum = 1:2,
  method = "gompertz",
  year_mode = "yearly",
  verbose = FALSE
)

det1 <- dm.detrend.fit(fit1)
head(det1$detrended_data)
head(det1$parameters)

fit2 <- dm.growth.fit.double(
  df = gf_nepa17,
  TreeNum = 1,
  method = "gompertz",
  year_mode = "yearly",
  verbose = FALSE
)

det2 <- dm.detrend.fit(fit2)
head(det2$detrended_long)
```

`dm.fit.gompertz` *Fitting gompertz function on annual dendrometer data*

Description

This function modells the annual growth of dendrometer data using gompertz function.

Usage

```
dm.fit.gompertz(
  df,
  CalYear,
  TreeNum,
  f_derivative = F,
```

```
  start = list(b = 0.5, k = 0.005),
  verbose = FALSE
)
```

Arguments

df	dataframe with first column containing date and time in the format yyyy-mm-dd HH:MM:SS and the dendrometer data in following columns.
CalYear	numeric for year of calculation. If df has more than one year, assigning CalYear truncates the data of only that year.
TreeNum	numerical value indicating the tree to be analysed. E.g. '1' refers to the first dendrometer data column in *df*.
f_derivative	logical if yes returns first derivative of gompertz curve.
start	A named list of start values for fitting the Gompertz curve (passed to nlsLM). Default is list(b = 0.5, k = 0.005).
verbose	logical if TRUE also returns the optimized parameters. Default is FALSE.

Value

A data frame with the modelled dendrometer series. If verbose = TRUE, returns a list with two data frames. The fitted curve and parameters.

Examples

```
library(dendRoAnalyst)
data(gf_nepa17)
gomp_fitted<-dm.fit.gompertz(df=gf_nepa17, TreeNum = 1, CalYear=2017)
head(gomp_fitted,10)
```

---

dm.growth.evaluate *Compare dendrometer growth-fitting methods using fit statistics*

Description

Fits one or more growth models to dendrometer series and returns a compact table of evaluation statistics only.

This function is intended for method comparison rather than data extraction. It runs the selected fitting methods, aligns observed and fitted values on all non-missing observation days, and calculates goodness-of-fit measures for each fitted series and vegetation-season fit.

Supported single-curve methods are '"gam"', '"gompertz"', '"logistic"', '"richards"', '"loess"', and '"spline"', which are fitted using [dm.growth.fit()]. Supported bimodal methods are '"double_gompertz"' and '"double_richards"', which are fitted using [dm.growth.fit.double()].

The function returns only an evaluation table and does not return fitted curves, processed data, or parameter objects. It is therefore useful for comparing alternative fitting methods across series, years, or sites before selecting a final modeling approach.

Usage

```
dm.growth.evaluate(
  df,
  TreeNum = "all",
  methods = c("gam", "gompertz", "logistic", "richards", "loess", "spline",
    "double_gompertz", "double_richards"),
  years = "all",
  year_mode = c("yearly", "pooled"),
  fit_GRO = TRUE,
  site_mode = c("NH", "SH", "CS"),
  custom_veg_season = c(275, 274),
  growth_fraction = c(0.1, 0.9),
  min_unique_growth = 10,
  rate_threshold_fraction = 0.1,
  peak_separation_min = 10,
  valley_ratio_max = 0.4,
  min_relative_peak = 0.05,
 start_value_gompertz_parameters = list(a = NA_real_, b = NA_real_, k = NA_real_),
 start_value_richards_parameters = list(a = NA_real_, k = NA_real_, t0 = NA_real_, v =
    1),
 start_value_double_gompertz_parameters = list(a = NA_real_, k = NA_real_, t0 =
    NA_real_, a2 = NA_real_, k2 = NA_real_, t02 = NA_real_),
 start_value_double_richards_parameters = list(a = NA_real_, k = NA_real_, t0 =
    NA_real_, v = 1, a2 = NA_real_, k2 = NA_real_, t02 = NA_real_, v2 = 1),
  loess_span = 0.2,
  spline_df = 10,
  verbose = TRUE
)
```

Arguments

df	A data frame or tibble. The first column must be a time stamp and the remaining selected columns must be numeric dendrometer series.
TreeNum	Either ‘"all"‘, a numeric vector selecting dendrometer series by position, or a character vector of series names.
methods	Character vector of fitting methods to evaluate.
years	Either ‘"all"‘ or a character vector of vegetation-season labels.
year_mode	Either ‘"yearly"‘ or ‘"pooled"‘.
fit_GRO	Logical. If ‘TRUE‘, convert processed daily series to cumulative growth before fitting.
site_mode	Vegetation season definition. One of ‘"NH"‘, ‘"SH"‘, or ‘"CS"‘.
custom_veg_season	Numeric vector of length 2 defining a custom season in day-of-year coordinates when ‘site_mode = "CS"‘.
growth_fraction	Numeric vector of length 2 used by the fitting functions to estimate cumulative-growth timing.

`min_unique_growth`
: Minimum number of unique cumulative-growth values required before a fit is attempted.

`rate_threshold_fraction`
: Numeric scalar between 0 and 1. Passed to the fitting functions where supported.

`peak_separation_min`
: Minimum peak separation in days for [dm.growth.fit.double()].

`valley_ratio_max`
: Maximum allowed valley-to-peak ratio for [dm.growth.fit.double()].

`min_relative_peak`
: Minimum relative peak height for [dm.growth.fit.double()].

`start_value_gompertz_parameters`
: Optional starting values for Gompertz fits.

`start_value_richards_parameters`
: Optional starting values for logistic and Richards fits.

`start_value_double_gompertz_parameters`
: Optional starting values for double-Gompertz fits.

`start_value_double_richards_parameters`
: Optional starting values for double-Richards fits.

`loess_span`
: Span used when ‘method = "loess"‘.

`spline_df`
: Degrees of freedom used when ‘method = "spline"‘.

`verbose`
: Logical. If ‘TRUE‘, prints progress messages.

Details

Evaluation statistics are calculated by comparing observed daily values against fitted daily values on all days where observed values are available.

The returned metrics have the following interpretation:

- ‘rmse‘: root mean squared error. Smaller values indicate closer fit.
- ‘mae‘: mean absolute error. Smaller values indicate closer fit and are less sensitive to large deviations than RMSE.
- ‘bias‘: mean fitted minus observed value. Positive values indicate overestimation and negative values indicate underestimation.
- ‘r2‘: coefficient of determination, computed from residual and total sums of squares. Larger values indicate better fit.
- ‘correlation‘: Pearson correlation between observed and fitted values. Larger values indicate stronger agreement in shape.
- ‘nrmse‘: normalized RMSE, calculated as RMSE divided by the observed value range. This allows comparison across series with different magnitudes.
- ‘rss‘: residual sum of squares.
- ‘aic_approx‘ and ‘bic_approx‘: approximate information criteria based on RSS and an effective parameter count rather than the exact likelihood returned by each modeling function.

The columns ‘aic_approx‘ and ‘bic_approx‘ are labeled as approximate because they are not extracted from the original model objects through a unified likelihood interface. Instead, they are computed from the residual sum of squares and an effective number of fitted parameters. They are therefore most useful for relative comparison among methods fitted to the same dataset.

Negative values of ‘aic_approx‘ or ‘bic_approx‘ are not an error. They occur naturally when the average residual variance is smaller than 1, because the logarithmic term in the information-criterion formula becomes negative. In practice, the absolute sign is not important; only differences among methods fitted to the same data should be interpreted.

For methods with flexible smoothness, such as GAM or spline, the effective parameter count is approximate. Consequently, ‘aic_approx‘ and ‘bic_approx‘ should be treated as heuristic ranking measures rather than exact likelihood-based criteria.

Value

A tibble with one row per fitted series and fit, containing:

method Fitting method used for the row.

series Series name.

fit_id Vegetation-season label or ‘"pooled"‘.

n_compare Number of observed days used for comparison.

rmse Root mean squared error.

mae Mean absolute error.

bias Mean fitted minus observed value.

r2 Coefficient of determination.

correlation Pearson correlation between observed and fitted values.

nrmse RMSE divided by the observed range.

rss Residual sum of squares.

aic_approx Approximate AIC based on RSS and effective parameter count.

bic_approx Approximate BIC based on RSS and effective parameter count.

k_effective Effective parameter count used in the approximate information criteria.

converged Logical convergence flag returned by the fitting function.

model_note Model note, warning, or error message returned by the fitting function.

See Also

[dm.growth.fit()], [dm.growth.fit.double()]

`dm.growth.fit` *Fit dendrometer growth curves by vegetation season*

Description

Fits cumulative growth curves to daily dendrometer series using one of several supported methods: generalized additive model `"gam"`, Gompertz `"gompertz"`, logistic `"logistic"`, Richards `"richards"`, local regression `"loess"`, or smoothing spline `"spline"`.

The function:

- resamples dendrometer data to daily maxima using [dendro.resample()],
- assigns each daily observation to a vegetation season,
- optionally converts daily stem-size values to cumulative growth with `fit_GRO = TRUE`,
- fits one curve per series and season with `year_mode = "yearly"` or one pooled curve across retained seasons with `year_mode = "pooled"`,
- estimates growing-season timing from the fitted cumulative-growth curve,
- additionally identifies active-growth timing from the fitted growth-rate curve.

For Gompertz, logistic, and Richards fits, the asymptote parameter `a`, representing maximum seasonal growth, can optionally be fixed to the observed maximum cumulative growth for that series and vegetation season. This is controlled by `fix_a_to_observed_max`.

Usage

```
dm.growth.fit(
  df,
  TreeNum = "all",
  method = c("gam", "gompertz", "logistic", "richards", "loess", "spline"),
  years = "all",
  year_mode = c("yearly", "pooled"),
  fit_GRO = TRUE,
  site_mode = c("NH", "SH", "CS"),
  custom_veg_season = c(275, 274),
  growth_fraction = c(0.1, 0.9),
  min_unique_growth = 10,
  rate_threshold_fraction = 0.1,
  fix_a_to_observed_max = FALSE,
  fixed_a_multiplier = 1,
 start_value_gompertz_parameters = list(a = NA_real_, b = NA_real_, k = NA_real_),
 start_value_richards_parameters = list(a = NA_real_, k = NA_real_, t0 = NA_real_, v =
    1),
  loess_span = 0.2,
  spline_df = 10,
  verbose = TRUE
)
```

Arguments

df	A data frame or tibble. The first column must be a timestamp Date, POSIXct, or parseable date-time string. All selected remaining columns must be numeric dendrometer series.
TreeNum	Either "all" to use all dendrometer series, a numeric vector selecting dendrometer columns by position, or a character vector with series names.
method	Fitting method. One of "gam", "gompertz", "logistic", "richards", "loess", or "spline".
years	Either "all" to fit all available vegetation seasons, or a character vector of season labels to retain.
year_mode	Either "yearly" to fit one curve per vegetation season, or "pooled" to fit one pooled curve across all retained seasons.
fit_GRO	Logical. If TRUE, processed daily series are converted to cumulative growth using a cumulative maximum within each vegetation season.
site_mode	Vegetation season definition. One of "NH", "SH", or "CS".
custom_veg_season	Numeric vector of length two giving the start and end day-of-year for custom vegetation seasons in "CS" mode. Cross-year seasons are supported, for example c(275, 274).
growth_fraction	Numeric vector of length two giving the lower and upper fractions of final fitted seasonal growth used to define cumulative growth onset and cessation, for example c(0.1, 0.9).
min_unique_growth	Minimum number of unique non-missing cumulative growth values required before a fit is attempted.
rate_threshold_fraction	Numeric scalar between 0 and 1. Active-growth dates are derived from the fitted growth-rate curve as the first and last days where fitted growth rate exceeds this fraction of the peak fitted growth rate.
fix_a_to_observed_max	Logical. If TRUE and method = "gompertz", "logistic", or "richards", the asymptote parameter a, representing maximum seasonal growth, is fixed to the observed maximum cumulative growth of the corresponding series and vegetation season. This is most appropriate with year_mode = "yearly". For year_mode = "pooled", one pooled maximum is used.
fixed_a_multiplier	Numeric multiplier applied to the observed maximum when fix_a_to_observed_max = TRUE. The default is 1, meaning a is fixed exactly to the observed seasonal maximum. Values slightly above 1, for example 1.01 or 1.05, allow the fitted asymptote to sit just above the observed maximum.
start_value_gompertz_parameters	Optional named list with starting values for Gompertz fits. Supported names are a, b, and k. If fix_a_to_observed_max = TRUE, supplied a is ignored.

`start_value_richards_parameters`
: Optional named list with starting values for Richards and logistic fits. Supported names are `a`, `k`, `t0`, and `v`. If `fix_a_to_observed_max = TRUE`, supplied `a` is ignored.

`loess_span`
: Span used when `method = "loess"`.

`spline_df`
: Degrees of freedom used when `method = "spline"`.

`verbose`
: Logical. If `TRUE`, prints a short completion message.

Details

The first column of `df` is treated as the time column and renamed to `"TIME"` internally. If it is not already a date-time object, the function attempts to parse it using [lubridate::parse_date_time()].

`growth_start_*` and `growth_end_*` are based on cumulative fitted growth. `rate_start_*` and `rate_end_*` are based on the fitted growth-rate curve.

For Gompertz, logistic, and Richards models, `a` is the upper asymptote. When `fix_a_to_observed_max = TRUE`, this parameter is not estimated by nonlinear least squares. Instead, it is fixed to:

$$a = max(y_{obs}) \times fixed\_a\_multiplier$$

where y_{obs} is the observed cumulative growth for that series and vegetation season.

Value

An object of class `"dm_growth_fit"` with elements:

call The matched function call.

original_daily_data Raw daily dendrometer data after resampling to daily maxima and assigning vegetation seasons, but before centering and optional cumulative-growth transformation.

processed_data Daily processed data used for fitting.

fitted_data Daily fitted values on the full vegetation-season grid.

fit_statistics Table with fit-level statistics and estimated timing.

fit_parameters Table with fit-level model parameters and convergence information.

season_table Vegetation seasons retained for fitting.

See Also

[summary.dm_growth_fit()], [print.dm_growth_fit()]

Examples

```
# fit <- dm.growth.fit(
#   df = dendro_data,
#   TreeNum = "all",
#   method = "gompertz",
#   year_mode = "yearly",
#   fit_GRO = TRUE,
#   fix_a_to_observed_max = TRUE
```

```
# )

# fit2 <- dm.growth.fit(
#   df = dendro_data,
#   TreeNum = "all",
#   method = "richards",
#   year_mode = "yearly",
#   fit_GRO = TRUE,
#   fix_a_to_observed_max = TRUE,
#   fixed_a_multiplier = 1.02
# )
```

`dm.growth.fit.double` *Fit bimodal dendrometer growth curves by vegetation season*

Description

Fits bimodal cumulative growth curves to daily dendrometer series using either a double-Gompertz or double-Richards model.

Overall growing-season timing is derived from the fitted cumulative-growth curve using `growth_fraction`. Pulse-specific timing is derived from the derivative of the fitted curve using a pulse-specific relative rate threshold given by `rate_threshold_fraction`.

Pulse-specific timing is returned as `NA` when the fitted curve does not show a convincing two-pulse pattern according to derivative-based criteria.

If `fallback_to_single = TRUE` and no convincing two-pulse pattern is found, the function refits a corresponding single growth curve and returns that fit instead. In that case, overall season timing is still returned, but pulse-specific timing remains `NA`.

For double-Gompertz and double-Richards fits, the total asymptote can optionally be fixed to the observed maximum cumulative growth of the corresponding series and vegetation season. In the double-curve case, this means `a + a2` is fixed, while the relative contribution of the first and second pulse is estimated.

Usage

```
dm.growth.fit.double(
  df,
  TreeNum = "all",
  method = c("gompertz", "richards"),
  years = "all",
  year_mode = c("yearly", "pooled"),
  fit_GRO = TRUE,
  site_mode = c("NH", "SH", "CS"),
  custom_veg_season = c(275, 274),
  growth_fraction = c(0.1, 0.9),
  min_unique_growth = 10,
```

```
  rate_threshold_fraction = 0.1,
  peak_separation_min = 10,
  valley_ratio_max = 0.4,
  min_relative_peak = 0.05,
  fallback_to_single = TRUE,
  fix_a_to_observed_max = FALSE,
  fixed_a_multiplier = 1,
  start_value_double_gompertz_parameters = list(a = NA_real_, k = NA_real_, t0 =
    NA_real_, a2 = NA_real_, k2 = NA_real_, t02 = NA_real_),
  start_value_double_richards_parameters = list(a = NA_real_, k = NA_real_, t0 =
    NA_real_, v = 1, a2 = NA_real_, k2 = NA_real_, t02 = NA_real_, v2 = 1),
  verbose = TRUE
)
```

Arguments

df	A data frame or tibble. The first column must be a time stamp Date, POSIXct, or parseable date-time string. Remaining selected columns must be numeric dendrometer series.
TreeNum	Either "all" to use all dendrometer series, a numeric vector selecting dendrometer columns by position, or a character vector of column names.
method	Double-curve fitting method. One of "gompertz" or "richards".
years	Either "all" to fit all available vegetation seasons, or a character vector of season labels to retain.
year_mode	Either "yearly" to fit one curve per vegetation season, or "pooled" to fit one pooled curve across all retained seasons.
fit_GRO	Logical. If TRUE, processed daily series are converted to cumulative growth using a cumulative maximum within each vegetation season.
site_mode	Vegetation season definition. One of "NH", "SH", or "CS".
custom_veg_season	Numeric vector of length two giving the start and end day-of-year for custom vegetation seasons in "CS" mode.
growth_fraction	Numeric vector of length two giving the lower and upper fractions of final fitted seasonal growth used to define overall growing-season onset and cessation.
min_unique_growth	Minimum number of unique non-missing cumulative growth values required before a fit is attempted.
rate_threshold_fraction	Numeric scalar between 0 and 1. Pulse start and end are defined as the first and last days where fitted growth rate exceeds this fraction of the pulse-specific peak fitted growth rate.
peak_separation_min	Minimum number of days separating the two fitted derivative peaks required to classify a fit as truly two-pulse.

`valley_ratio_max`
: Maximum allowed ratio between the valley rate and the weaker of the two derivative peaks. Smaller values require a deeper valley between pulses.

`min_relative_peak`
: Minimum relative height, expressed as a fraction of the global derivative maximum, for a local derivative peak to be considered.

`fallback_to_single`
: Logical. If `TRUE`, and the fitted double curve does not show a convincing two-pulse pattern according to derivative-based criteria, the function refits a corresponding single growth curve `"gompertz"` or `"richards"` and returns that fit instead.

`fix_a_to_observed_max`
: Logical. If `TRUE`, the total asymptote of the double-growth curve is fixed to the observed maximum cumulative growth for that series and vegetation season. For double-Gompertz and double-Richards models, this means `a + a2` is fixed, while the relative contribution of the two pulses is estimated. This is most appropriate with `year_mode = "yearly"`. For `year_mode = "pooled"`, one pooled maximum is used.

`fixed_a_multiplier`
: Numeric multiplier applied to the observed maximum when `fix_a_to_observed_max = TRUE`. The default is `1`, meaning the total asymptote is fixed exactly to the observed seasonal maximum. Values slightly above 1, for example `1.01` or `1.05`, allow the fitted asymptote to sit slightly above the observed maximum.

`start_value_double_gompertz_parameters`
: Optional named list of starting values for the double-Gompertz fit. Supported names are `a`, `k`, `t0`, `a2`, `k2`, and `t02`. If `fix_a_to_observed_max = TRUE`, supplied `a` and `a2` are used only to initialize the relative pulse contribution.

`start_value_double_richards_parameters`
: Optional named list of starting values for the double-Richards fit. Supported names are `a`, `k`, `t0`, `v`, `a2`, `k2`, `t02`, and `v2`. If `fix_a_to_observed_max = TRUE`, supplied `a` and `a2` are used only to initialize the relative pulse contribution.

`verbose`
: Logical. If `TRUE`, prints a short completion message.

Details

The second pulse is constrained to occur after the first pulse, which improves numerical stability and reduces label switching between the two components.

For double-Gompertz and double-Richards models, the total seasonal asymptote is:

$$a_{total} = a + a2$$

When `fix_a_to_observed_max = TRUE`, the function fits:

$$a + a2 = max(y_{obs}) \times fixed\_a\_multiplier$$

where y_{obs} is the observed cumulative growth of the selected dendrometer series and vegetation season.

The fitted pulse contributions are still returned as `a` and `a2`, and their sum should equal `fixed_a_value`, apart from small numerical rounding.

Non-applicable parameters are returned as `NA`. For example, `b` and `b2` are relevant for double-Gompertz, while `v` and `v2` are relevant for double-Richards.

Value

An object of class `"dm_growth_fit"` with elements:

call The matched function call.

original_daily_data Raw daily dendrometer data after resampling to daily maxima and assigning vegetation seasons, but before centering and optional cumulative-growth transformation.

processed_data Daily processed data used for fitting.

fitted_data Daily fitted values on the full vegetation-season grid.

fit_statistics Fit-level statistics and estimated timing.

fit_parameters Fit-level model parameters and convergence information.

season_table Vegetation seasons retained for fitting.

See Also

[dm.growth.fit()], [summary.dm_growth_fit()], [print.dm_growth_fit()]

Examples

```
# Double-Gompertz with total seasonal asymptote fixed
# fit_gomp <- dm.growth.fit.double(
#   df = dendro_data,
#   TreeNum = "all",
#   method = "gompertz",
#   year_mode = "yearly",
#   fit_GRO = TRUE,
#   fix_a_to_observed_max = TRUE
# )

# Double-Richards with the total asymptote set 2 percent above observed max
# fit_rich <- dm.growth.fit.double(
#   df = dendro_data,
#   TreeNum = "all",
#   method = "richards",
#   year_mode = "yearly",
#   fit_GRO = TRUE,
#   fix_a_to_observed_max = TRUE,
#   fixed_a_multiplier = 1.02
# )
```

`dm.na.interpolation` *Detection and interpolation of missing values in dendrometer data*

Description

Detects missing timestamps (based on provided resolution), inserts rows with `NA`, and optionally interpolates missing values using either cubic spline interpolation (`na.spline`) or seasonal decomposition interpolation (`na.interp`).

Optionally, the function can test the reliability of interpolation for increasing gap lengths (e.g., 6 hours to 3 days) using real data. This allows the user to assess how well a given interpolation method recovers missing values over different durations.

Usage

```
dm.na.interpolation(
  df,
  resolution,
  fill = FALSE,
  method = "spline",
  assess = FALSE,
  assess_lengths_hours = seq(6, 72, by = 6),
  assess_samples_per_length = 10,
  assess_buffer_hours = 6,
  assess_seed = NULL,
  verbose = FALSE
)
```

Arguments

`df`	A data frame with the first column as datetime (`"yyyy-mm-dd HH:MM:SS"`, or POSIXct/Date), followed by dendrometer values.
`resolution`	Integer. Temporal resolution in minutes (e.g., 60 for hourly data).
`fill`	Logical. If `TRUE`, missing values will be interpolated.
`method`	Character. Interpolation method (only used if `fill = TRUE`): • `"spline"` — cubic spline (`zoo::na.spline`). • `"seasonal"` — seasonal decomposition (`forecast::na.interp`).
`assess`	Logical. If `TRUE`, run a simulation-based test to evaluate how well interpolation performs at different gap lengths (on existing data).
`assess_lengths_hours`	Integer vector. The gap durations (in hours) to test during assessment. Default: `seq(6, 72, by = 6)`.
`assess_samples_per_length`	Integer. Number of artificial gaps per gap length per series. Default: `10`.

`assess_buffer_hours`
: Integer. Minimum number of hours of valid data required before and after the artificial gap to allow valid assessment. Default: `6`.

`assess_seed`
: Integer or `NULL`. Random seed to ensure reproducibility of sampling windows. Default: `NULL`.

`verbose`
: Logical. If `TRUE`, prints messages during filling and testing.

Details

The assessment simulates interpolation over artificial gaps of different durations. For each dendrometer series and each gap length:

1. Random windows (with clean data) are selected.
2. Data in the window is temporarily set to `NA`.
3. The gap is interpolated using the selected method.
4. The predicted values are compared to the original (true) values using:
   - MAPE — Mean Absolute Percentage Error.
   - MdAPE — Median Absolute Percentage Error.
   - RMSE_pct — Root Mean Square Error (%).
   - Bias_pct — Mean Percentage Error (%).
   - Max_diff_abs — Mean absolute difference in daily maximum.
   - Min_diff_abs — Mean absolute difference in daily minimum.
   - Time_max_diff_h — Mean absolute difference in time of daily maximum (hours).
   - Time_min_diff_h — Mean absolute difference in time of daily minimum (hours).

Value

A list of class `"dm_na_interpolation"` with components:

`$data` Data frame containing the original or gap-filled series.

`$gap_info` Table describing timestamp gaps and internal NA runs.

`$assessment` A tibble summarizing interpolation accuracy for each series and gap length, or `NULL` when `assess = FALSE`.

`$filled` Logical indicating whether interpolation was performed.

`$assessed` Logical indicating whether interpolation assessment was performed.

`$method` Interpolation method used.

`$resolution` Temporal resolution in minutes.

Examples

```
library(dendRoAnalyst)
data(nepa17)

#res0 <- dm.na.interpolation(nepa17[1:1000, ], resolution = 60)
#plot(res0)
#plot(res0, type = "gaps")
```

```
#res1 <- dm.na.interpolation(
#  nepa17[1:1000, ],
#  resolution = 60,
#  fill = TRUE,
#  method = "spline"
#)
#plot(res1)
#plot(res1, type = "gaps")
#plot(res1, type = "interpolation", original = nepa17[1:1000, ])

#res2 <- dm.na.interpolation(
#  nepa17[1:1000, ],
#  resolution = 60,
#  fill = TRUE,
#  method = "seasonal",
#  assess = TRUE
# )
#plot(res2)
#plot(res2, type = "assessment", metric = "MdAPE")
```

`dm_add_climate` *Add climate information to dendrometer outputs*

Description

A single helper function that adds climate information to outputs from `daily.data()`, `phase.zg()`, or `phase.sc()`.

Depending on the class of `x` and the selected `scale`, the function:

- joins daily climate to `daily.data()` output
- joins climate summaries to phase windows in `ZG_cycle` or `SC_cycle`
- joins subdaily climate features to point-level `ZG_phase` or `SC_phase`

Usage

```
dm_add_climate(
  x,
  clim,
  scale = c("auto", "daily", "phase", "subdaily"),
  mean_vars = NULL,
  min_vars = NULL,
  max_vars = NULL,
  sum_vars = NULL,
  median_vars = NULL,
  lag_vars = NULL,
```

```
  lagmean_vars = NULL,
  lagsum_vars = NULL,
  lag_days = c(1, 3, 7),
  sub_mean_vars = NULL,
  sub_sum_vars = NULL,
  sub_lag_vars = NULL,
  roll_hours = c(3, 6, 24),
  lag_hours = c(1, 3, 6, 24),
  suffix = "_phase"
)
```

Arguments

`x`	Object returned by `daily.data()`, `phase.zg()`, or `phase.sc()`.
`clim`	Climate input. This can be: • a standardized object returned by `read.climate()` • a raw climate data frame • a valid climate file path readable by `read.climate()` • a daily climate table returned by `dm_daily_clim()` • a subdaily climate-feature table returned by `dm_subdaily_clim()`
`scale`	Character string controlling how climate is attached. One of `"auto"`, `"daily"`, `"phase"`, or `"subdaily"`. • For `daily_output`, only `"daily"` is supported. • For `ZG_output` and `SC_output`, `"phase"` adds climate summaries to `ZG_cycle`/`SC_cycle`, while `"subdaily"` adds timestamp-level climate to `ZG_phase`/`SC_phase`.
`mean_vars`, `min_vars`, `max_vars`, `sum_vars`, `median_vars`	Climate variables to summarize by mean, minimum, maximum, sum, or median. These are used by `dm_daily_clim()` and `dm_join_phase_clim()`.
`lag_vars`, `lagmean_vars`, `lagsum_vars`	Variables used by `dm_daily_clim()` to build lagged and antecedent daily climate features.
`lag_days`	Integer vector of lag/antecedent windows in days for `dm_daily_clim()`.
`sub_mean_vars`, `sub_sum_vars`, `sub_lag_vars`	Variables used by `dm_subdaily_clim()` to build rolling and lagged subdaily features.
`roll_hours`, `lag_hours`	Numeric vectors of rolling-window and lag sizes in hours for `dm_subdaily_clim()`.
`suffix`	Suffix appended to climate summaries added by `dm_join_phase_clim()`.

Value

The same biological object with climate information added.

Examples

```
data(nepa17)
data(gf_nepa17)
data(ktm_clim_hourly)

# daily.data() output + daily climate
dd <- daily.data(df = nepa17[1:1000, ], TreeNum = 1)
dd_clim <- dm_add_climate(
  dd,
  ktm_clim_hourly,
  scale = "daily",
  mean_vars = c("temp", "VPD", "RH"),
  max_vars  = c("temp", "VPD"),
  sum_vars  = c("prec")
)
head(dd_clim)

# phase.zg() output + phase-window climate
zg <- phase.zg(df = gf_nepa17[1:800, ], TreeNum = 1)
zg_clim <- dm_add_climate(
  zg,
  ktm_clim_hourly,
  scale = "phase",
  mean_vars = c("temp", "VPD", "RH"),
  max_vars  = c("temp", "VPD"),
  sum_vars  = c("prec")
)
head(zg_clim$ZG_cycle)

# phase.sc() output + point-level subdaily climate
sc <- phase.sc(df = gf_nepa17[1:800, ], TreeNum = 1, smoothing = 12)
sc_point <- dm_add_climate(
  sc,
  ktm_clim_hourly,
  scale = "subdaily",
  sub_mean_vars = c("temp", "VPD", "RH"),
  sub_sum_vars  = c("prec"),
  sub_lag_vars  = c("temp", "VPD", "RH"),
  roll_hours = c(1, 3, 6, 24),
  lag_hours  = c(1, 3, 6, 24)
)
head(sc_point$SC_phase)
```

`dm_daily_clim` *Daily climate summaries for dendrometer analyses*

Description

Computes daily climate summaries from climate time series so they can be related to daily dendrometer summaries from `daily.data()`.

The input can be a standardized climate object returned by `read.climate()`, a raw data frame, or a valid file path accepted by `read.climate()`.

In addition to same-day climate summaries, the function can also compute lagged and antecedent daily climate features from the summarized daily series:

- lagged values (e.g. previous 1 or 3 days)
- antecedent means over previous n days
- antecedent sums over previous n days

Usage

```
dm_daily_clim(
  clim_df,
  mean_vars = NULL,
  min_vars = NULL,
  max_vars = NULL,
  sum_vars = NULL,
  median_vars = NULL,
  lag_vars = NULL,
  lagmean_vars = NULL,
  lagsum_vars = NULL,
  lag_days = c(1, 3, 7)
)
```

Arguments

`clim_df`	Climate input. This can be: • a standardized object returned by `read.climate()` • a raw data frame with a time column in the first column or in a column named `TIME` • a valid file path readable by `read.climate()`
`mean_vars`	Character vector of variables to summarize by mean.
`min_vars`	Character vector of variables to summarize by minimum.
`max_vars`	Character vector of variables to summarize by maximum.
`sum_vars`	Character vector of variables to summarize by sum.
`median_vars`	Character vector of variables to summarize by median.
`lag_vars`	Character vector of summarized daily climate columns for which simple lagged values should be computed, e.g. `c("VPD_max", "SWC_mean")`.
`lagmean_vars`	Character vector of summarized daily climate columns for which antecedent means should be computed, e.g. `c("Tair_mean", "VPD_mean")`.
`lagsum_vars`	Character vector of summarized daily climate columns for which antecedent sums should be computed, e.g. `c("P_sum", "Rad_sum")`.
`lag_days`	Integer vector giving lag/antecedent window sizes in days, e.g. `c(1, 3, 7)`.

Details

Lagged and antecedent features are calculated from the already summarized daily climate columns. For example, if `VPD` is included in `max_vars`, the daily summary column will be `VPD_max`. If this column is listed in `lag_vars` and `lag_days = 1`, then the additional column `VPD_max_lag_1d` is created.

Antecedent means and sums exclude the current day. For example:

$$x\_lagmean\_3d(t) = mean(x_{t-3}, x_{t-2}, x_{t-1})$$

$$x\_lagsum\_7d(t) = sum(x_{t-7}, \ldots, x_{t-1})$$

Value

A tibble of class `"daily_clim"` with one row per day.

Examples

```
data(ktm_clim_hourly)
clim_day <- dm_daily_clim(
  ktm_clim_hourly,
  mean_vars = c("temp", "VPD", "RH"),
  max_vars  = c("VPD"),
  sum_vars  = c("prec"),
  lag_vars = c("VPD_max", "temp_mean"),
  lagmean_vars = c("temp_mean", "VPD_mean", "RH_mean"),
  lagsum_vars = c("prec_sum"),
  lag_days = c(1, 3, 7)
)
head(clim_day, 5)
```

---

dm_epoch_extract *Build an epoch table around event times*

Description

Builds a long-format epoch table by extracting climate values before and after event times.

The optional `Year` and `DOY` arguments allow the user to restrict which events are included in the superposed epoch analysis. The full climate table is retained for lag extraction so that values before and after selected events are still available.

Usage

```
dm_epoch_extract(
  events,
  clim,
  vars = "all",
  lag_before = 24,
  lag_after = 24,
  step = 1,
  unit = c("hours", "days", "mins"),
  agg_fun = "mean",
  var_funs = NULL,
  Year = NULL,
  DOY = NULL,
  restrict_null_to_window = TRUE
)
```

Arguments

events	Output of [dm_event_times()] or a data frame containing an event_time column.
clim	Climate data frame. The first column must contain timestamps.
vars	Climate variables to use, or "all".
lag_before	Number of lag steps before the event.
lag_after	Number of lag steps after the event.
step	Step size in units.
unit	One of "hours", "days", or "mins".
agg_fun	Aggregation function applied to all climate variables when var_funs is not supplied.
var_funs	Optional named vector or named list of aggregation functions for each variable.
Year	Optional numeric vector of calendar years to include in the SEA. Events whose event_time falls outside these years are removed before epoch extraction.
DOY	Optional numeric vector of length one or two giving the day-of-year window to include in the SEA. If length one, only that DOY is retained. If length two, the inclusive range is retained. Cross-year DOY windows are supported, for example DOY = c(300, 60).
restrict_null_to_window	Logical. If TRUE, null anchor times in [dm_epoch_test()] are sampled only from the same Year/DOY window. The full climate table is still retained for extracting lag_before and lag_after windows. Default is TRUE.

Value

An object of class c("dm_epoch", "dm_epoch_extract").

Examples

```
# events <- dm_event_times(zg, event = "GRO_start")
# ep <- dm_epoch_extract(
#   events = events,
#   clim = climate,
#   vars = c("Rain", "temp", "vpd"),
#   Year = c(2022, 2023, 2024),
#   DOY = c(100, 300),
#   lag_before = 24,
#   lag_after = 24,
#   unit = "hours"
# )
```

`dm_epoch_test` *Test superposed epoch composites against a null distribution*

Description

Tests observed superposed epoch composites against empirical null distributions generated by randomly sampled anchor times.

If the input object was created with `Year` and/or `DOY` filters in [dm_epoch_extract()] and `restrict_null_to_window = TRUE`, null anchors are sampled from the same calendar-year and day-of-year window.

Usage

```
dm_epoch_test(
  x,
  statistic = c("mean", "median"),
  null = c("same_doy_window", "same_month", "random_time"),
  n_iter = 1000,
  doy_window = 15,
  match_hour = TRUE,
  match_minute = TRUE,
  conf_level = 0.95,
  seed = NULL
)
```

Arguments

`x`	Object returned by [dm_epoch_extract()].
`statistic`	Composite statistic. One of `"mean"` or `"median"`.
`null`	Null model. One of `"same_doy_window"`, `"same_month"`, or `"random_time"`.
`n_iter`	Number of null iterations.
`doy_window`	DOY window used for `null = "same_doy_window"`.

match_hour	Logical. For hourly or minute data, keep the same hour when possible.
match_minute	Logical. For minute data, keep the same minute when possible.
conf_level	Confidence level for null envelopes.
seed	Optional random seed.

Value

An object of class c("dm_epoch", "dm_epoch_test").

---

dm_event_climate *Extract climate at and before phase events*

Description

Extracts climate at event times and summarizes climate in windows preceding those events. Event times can be phase starts, phase ends, or Max.twd.time for phase.zg() output.

Usage

```
dm_event_climate(
  x,
  clim_df,
  event = c("phase_start", "phase_end", "max_twd"),
  phase = "all",
  windows = c(0, 1, 3, 6, 12, 24, 48),
  mean_vars = NULL,
  max_vars = NULL,
  min_vars = NULL,
  sum_vars = NULL,
  median_vars = NULL,
  instant_vars = NULL,
  instant_match = c("nearest", "previous", "next")
)
```

Arguments

x	Output of phase.zg() or phase.sc().
clim_df	Subdaily climate input. This can be a standardized climate object from read.climate(), a raw climate data frame, or a valid climate file path.
event	One of "phase_start", "phase_end", or "max_twd".
phase	Phase selector. For ZG_output: "all", "TWD", or "GRO". For SC_output: "all", "Shrinkage", "Expansion", or "Increment".
windows	Numeric vector of antecedent windows in hours, e.g. c(0, 1, 3, 6, 12, 24, 48). A value of 0 extracts climate at the event time.

`mean_vars`, `max_vars`, `min_vars`, `sum_vars`, `median_vars`
: Climate variables to summarize in antecedent windows.

`instant_vars`
: Climate variables to extract at the exact event time. If `NULL`, all selected variables are used.

`instant_match`
: Matching rule for event-time extraction: `"nearest"`, `"previous"`, or `"next"`.

Value

A tibble with one row per event and climate summaries in the requested antecedent windows.

Examples

```
data(gf_nepa17)
data(ktm_clim_hourly)

zg <- phase.zg(df = gf_nepa17[1:800, ], TreeNum = 1)

evt_zg <- dm_event_climate(
  zg,
  ktm_clim_hourly,
  event = "phase_start",
  phase = "all",
  windows = c(0, 1, 3, 6, 12, 24),
  mean_vars = c("temp", "VPD", "RH"),
  max_vars  = c("temp", "VPD"),
  sum_vars  = c("prec")
)

head(evt_zg)

evt_maxtwd <- dm_event_climate(
  zg,
  ktm_clim_hourly,
  event = "max_twd",
  windows = c(0, 1, 3, 6, 12, 24),
  mean_vars = c("temp", "VPD", "RH"),
  max_vars  = c("VPD"),
  sum_vars  = c("prec")
)

head(evt_maxtwd)
```

---

dm_event_climate_summary

Summarize event-based climate results

Description

Summarizes a climate variable extracted by `dm_event_climate()` by phase, event type, month, or month-of-year.

Usage

```
dm_event_climate_summary(
  event_data,
  climate_var,
  group_var = c("Phase", "event_type"),
  by = c("none", "month", "month_of_year", "year"),
  Year = NULL,
  DOY = NULL
)
```

Arguments

event_data	Output of dm_event_climate().
climate_var	Name of the climate-derived column to summarize.
group_var	Grouping variable. One of "Phase" or "event_type".
by	One of "none", "month", "month_of_year", or "year".
Year	Optional numeric year or vector of years for filtering.
DOY	Optional numeric vector of length 2 for filtering.

Value

A tibble of summary statistics.

Examples

```
data(gf_nepa17)
data(ktm_clim_hourly)

zg <- phase.zg(df = gf_nepa17[1:800, ], TreeNum = 1)
evt_zg <- dm_event_climate(
  zg,
  ktm_clim_hourly,
  event = "phase_start",
  windows = c(0, 3, 6, 12, 24),
  mean_vars = c("temp", "VPD", "RH"),
  max_vars  = c("VPD"),
  sum_vars  = c("prec")
)

smry <- dm_event_climate_summary(
  evt_zg,
  climate_var = "VPD_mean_prev_6h",
  group_var = "Phase",
  by = "month_of_year"
)

head(smry)
```

dm_event_climate_test *Test differences in event-based climate between groups*

Description

Tests whether an event-based climate variable differs between groups such as phases or event types.

Usage

```
dm_event_climate_test(
  event_data,
  climate_var,
  group_var = c("Phase", "event_type"),
  by = c("none", "month", "month_of_year", "year"),
  Year = NULL,
  DOY = NULL,
  method = c("auto", "anova", "kruskal", "t.test", "wilcox")
)
```

Arguments

event_data	Output of dm_event_climate().
climate_var	Name of the climate-derived column to test.
group_var	Grouping variable. One of "Phase" or "event_type".
by	One of "none", "month", "month_of_year", or "year".
Year	Optional numeric year or vector of years for filtering.
DOY	Optional numeric vector of length 2 for filtering.
method	One of "auto", "anova", "kruskal", "t.test", or "wilcox".

Value

A list with summary statistics, overall tests, and pairwise tests.

Examples

```
data(gf_nepa17)
data(ktm_clim_hourly)

zg <- phase.zg(df = gf_nepa17[1:800, ], TreeNum = 1)
evt_zg <- dm_event_climate(
  zg,
  ktm_clim_hourly,
  event = "phase_start",
  windows = c(0, 3, 6, 12, 24),
  mean_vars = c("temp", "VPD", "RH"),
  max_vars  = c("VPD"),
```

```
  sum_vars  = c("prec")
)

tst <- dm_event_climate_test(
  evt_zg,
  climate_var = "VPD_mean_prev_6h",
  group_var = "Phase",
  by = "month_of_year",
  method = "auto"
)

tst$summary
tst$overall_tests
head(tst$pairwise_tests)
```

`dm_event_times` *Extract event times from phase.zg() or phase.sc() output*

Description

Extracts event times from objects returned by `phase.zg()` or `phase.sc()`. The resulting table can be passed directly to [dm_epoch_extract()] for superposed epoch analysis.

Usage

```
dm_event_times(
  x,
  event = "all",
  phase = NULL,
  min_duration = NULL,
  min_magnitude = NULL,
  min_max_twd = NULL,
  remove_na_times = TRUE
)
```

Arguments

`x`	Object of class `"ZG_output"` or `"SC_output"`.
`event`	Event type to extract. Use `"all"` to return all supported events. For ZG output, supported events are `"TWD_start"`, `"TWD_end"`, `"GRO_start"`, `"GRO_end"`, `"MaxTWD"`, `"phase_start"`, and `"phase_end"`. For SC output, supported events are `"Shrinkage_start"`, `"Shrinkage_end"`, `"Expansion_start"`, `"Expansion_end"`, `"Increment_start"`, `"Increment_end"`, `"phase_start"`, and `"phase_end"`.
`phase`	Optional phase filter for generic events `"phase_start"` or `"phase_end"`. For ZG output, use `"TWD"` or `"GRO"`. For SC output, use `"Shrinkage"`, `"Expansion"`, or `"Increment"`.

min_duration	Optional minimum Duration_h filter.
min_magnitude	Optional minimum Magnitude filter.
min_max_twd	Optional minimum max.twd filter for ZG output.
remove_na_times	Logical. If TRUE, rows with missing event_time are removed.

Value

A tibble of events with class c("dm_events", ...).

---

dm_join_daily_clim *Join daily climate summaries to daily dendrometer statistics*

Description

Joins daily climate summaries to the output of daily.data() by calendar date.

The climate input can be:

- the output of dm_daily_clim()
- the output of read.climate()
- a raw climate data frame
- a valid climate file path readable by read.climate()

If the supplied climate input is not already a daily climate table, it will first be converted to daily summaries with dm_daily_clim() using default daily means for numeric variables.

Usage

```
dm_join_daily_clim(daily_obj, clim_daily)
```

Arguments

daily_obj	Output of daily.data() with class "daily_output".
clim_daily	Climate input. This can be a daily climate table, a standardized climate object returned by read.climate(), a raw climate data frame, or a valid climate file path.

Value

The daily.data() output with climate columns appended. The returned object has class c("daily_output_clim", "daily_output", ...).

Examples

```
data(nepa17)
data(ktm_clim_hourly)

dd <- daily.data(df = nepa17[1:1000, ], TreeNum = 1)

clim_std <- read.climate(
  ktm_clim_hourly,
  time_col = "TIME",
  vars = c("temp", "prec", "VPD", "RH"),
  verbose = FALSE
)

clim_day <- dm_daily_clim(
  clim_std,
  mean_vars = c("temp", "VPD", "RH"),
  max_vars  = c("temp", "VPD"),
  sum_vars  = c("prec")
)

dd_clim <- dm_join_daily_clim(dd, clim_day)
head(dd_clim)
```

`dm_join_phase_clim` *Join climate summaries to dendrometer phase windows*

Description

Summarizes climate conditions within each phase window of `phase.zg()` or `phase.sc()` output and appends those summaries to `ZG_cycle` or `SC_cycle`.

The climate input can be:

- the output of `read.climate()`
- a raw climate data frame
- a valid climate file path readable by `read.climate()`

Climate variables are summarized over each phase interval defined by the `Start` and `End` columns of the phase-cycle table.

Usage

```
dm_join_phase_clim(
  x,
  clim_df,
  mean_vars = NULL,
  min_vars = NULL,
```

```
  max_vars = NULL,
  sum_vars = NULL,
  median_vars = NULL,
  suffix = "_phase"
)
```

Arguments

x	Object returned by `phase.zg()` or `phase.sc()`.
clim_df	Climate input. This can be a standardized climate object returned by `read.climate()`, a raw climate data frame, or a valid climate file path.
mean_vars	Character vector of climate variables to summarize by mean within each phase window.
min_vars	Character vector of climate variables to summarize by minimum within each phase window.
max_vars	Character vector of climate variables to summarize by maximum within each phase window.
sum_vars	Character vector of climate variables to summarize by sum within each phase window.
median_vars	Character vector of climate variables to summarize by median within each phase window.
suffix	Character suffix appended to the generated climate summary columns. Default is `"_phase"`.

Value

The same phase object with climate summaries appended to `ZG_cycle` or `SC_cycle`. The returned object has class `c("ZG_output_clim", "ZG_output", ...)` or `c("SC_output_clim", "SC_output", ...)`.

Examples

```
data(gf_nepa17)
data(ktm_clim_hourly)

zg <- phase.zg(df = gf_nepa17[1:800, ], TreeNum = 1)

zg_clim <- dm_join_phase_clim(
  zg,
  ktm_clim_hourly,
  mean_vars = c("temp", "VPD", "RH"),
  max_vars  = c("temp", "VPD"),
  sum_vars  = c("prec")
)

head(zg_clim$ZG_cycle)
```

dm_join_subdaily_clim *Join subdaily climate features to point-level dendrometer output*

Description

Joins timestamp-level subdaily climate features to point-level output from phase.zg() or phase.sc().

The climate input can be:

- the output of dm_subdaily_clim()
- the output of read.climate()
- a raw climate data frame
- a valid climate file path readable by read.climate()

If the supplied climate input is not already a subdaily climate-feature table, the function attempts to standardize it with read.climate() and then joins by exact timestamp using the TIME column.

Usage

```
dm_join_subdaily_clim(x, clim_sub)
```

Arguments

x	Object returned by phase.zg() or phase.sc().
clim_sub	Climate input. This can be a subdaily climate-feature table, a standardized climate object returned by read.climate(), a raw climate data frame, or a valid climate file path.

Value

The same phase object with climate features appended to ZG_phase or SC_phase. The returned object has class c("ZG_output_clim", "ZG_output", ...) or c("SC_output_clim", "SC_output", ...).

Examples

```
data(gf_nepa17)
data(ktm_clim_hourly)

zg <- phase.zg(df = gf_nepa17[1:800, ], TreeNum = 1)

clim_sub <- dm_subdaily_clim(
  ktm_clim_hourly,
  mean_vars = c("temp", "VPD", "RH"),
  sum_vars  = c("prec"),
  lag_vars  = c("temp", "VPD", "RH"),
  roll_hours = c(1, 3, 6, 24),
  lag_hours  = c(1, 3, 6, 24)
)
```

```
zg_point_clim <- dm_join_subdaily_clim(zg, clim_sub)
head(zg_point_clim$ZG_phase)
```

`dm_plot_climate` *Plot climate attached to dendrometer outputs*

Description

General plotting function for dendrometer outputs with attached climate information. It works with objects of class `daily_output_clim`, `ZG_output_clim`, and `SC_output_clim`, usually produced by `dm_add_climate()`, `dm_join_daily_clim()`, `dm_join_phase_clim()`, or `dm_join_subdaily_clim()`.

The function can display one selected climate variable through time, by phase or day status, as boxplots or violin plots, or as a climate-response relationship.

Usage

```
dm_plot_climate(
  x,
  climate_var,
  metric_var = NULL,
  scale = c("auto", "daily", "cycle", "point"),
  type = c("timeseries", "boxplot", "violin", "both", "relation"),
  group_var = NULL,
  Year = NULL,
  DOY = NULL,
  point_alpha = 0.6,
  add_smooth = FALSE,
  temporal = NULL
)
```

Arguments

`x`	A climate-augmented dendrometer object.
`climate_var`	Character. Name of the climate variable to plot.
`metric_var`	Optional character. Biological response variable used when `type = "relation"`. If `NULL`, a suitable default is selected when possible.
`scale`	Character. Data level to use. One of `"auto"`, `"daily"`, `"cycle"`, or `"point"`.
`type`	Character. Plot type. One of `"timeseries"`, `"boxplot"`, `"violin"`, `"both"`, or `"relation"`.
`group_var`	Optional character. Grouping variable. If `NULL`, `Day_status` is used for daily outputs when available, and `Phases` is used for cycle- or point-level phase outputs.

Year	Optional numeric year or vector of years for subsetting.
DOY	Optional numeric vector of length 2 giving the day-of-year range.
point_alpha	Numeric. Alpha value for points.
add_smooth	Logical. If TRUE and type = "relation", a linear trend line is added.
temporal	Deprecated argument kept for compatibility with older examples. It is ignored.

Value

A ggplot2 object, returned invisibly.

Examples

```
data(gf_nepa17)
data(ktm_clim_hourly)

zg <- phase.zg(df = gf_nepa17[1:800, ], TreeNum = 1)
zg_clim <- dm_add_climate(
  zg,
  ktm_clim_hourly,
  scale = "phase",
  mean_vars = c("temp", "VPD", "RH"),
  max_vars  = c("temp", "VPD"),
  sum_vars  = c("prec")
)

dm_plot_climate(
  zg_clim,
  climate_var = "VPD_mean_phase",
  scale = "cycle",
  type = "boxplot"
)

plot(
  zg_clim,
  climate_var = "VPD_mean_phase",
  scale = "cycle",
  type = "boxplot"
)
```

---

dm_plot_climate_compare

Compare multiple climate variables attached to dendrometer outputs

---

Description

Plots and compares multiple climate variables attached to climate-augmented dendrometer outputs. It works with objects of class `daily_output_clim`, `ZG_output_clim`, and `SC_output_clim`.

Supported plot types include faceted time-series, grouped boxplots, violin plots, combined violin-boxplots, correlation heatmaps, and regression heatmaps.

Usage

```
dm_plot_climate_compare(
  x,
  climate_vars = NULL,
  numeric_vars = NULL,
  scale = c("auto", "daily", "cycle", "point"),
  type = c("timeseries", "boxplot", "violin", "both", "cor_heatmap",
    "regression_heatmap"),
  group_var = NULL,
  Year = NULL,
  DOY = NULL,
  box_scales = c("free_y", "fixed"),
  cor_method = c("pearson", "spearman", "kendall"),
  use_pairwise = TRUE,
  regression_stat = c("r.squared", "slope", "p.value"),
  point_alpha = 0.5,
  show_values = FALSE,
  temporal = NULL
)
```

Arguments

`x`	A climate-augmented dendrometer object.
`climate_vars`	Character vector of climate variables to compare. If `NULL`, all detected climate variables at the selected scale are used.
`numeric_vars`	Optional character vector of numeric variables used when `type = "cor_heatmap"` or `type = "regression_heatmap"`. If `NULL`, all numeric variables in the selected data are used.
`scale`	Character. Data level to use. One of `"auto"`, `"daily"`, `"cycle"`, or `"point"`.
`type`	Character. Plot type. One of `"timeseries"`, `"boxplot"`, `"violin"`, `"both"`, `"cor_heatmap"`, or `"regression_heatmap"`.
`group_var`	Optional character. Grouping variable. If `NULL`, `Day_status` is used for daily outputs when available, and `Phases` is used for phase outputs.
`Year`	Optional numeric year or vector of years for subsetting.
`DOY`	Optional numeric vector of length 2 giving the day-of-year range.
`box_scales`	Character. Facet scale behavior for grouped plots. One of `"free_y"` or `"fixed"`.
`cor_method`	Character. Correlation method. One of `"pearson"`, `"spearman"`, or `"kendall"`.
`use_pairwise`	Logical. If `TRUE`, pairwise complete observations are used in correlations.

`regression_stat`
: Character. Statistic shown in regression heatmaps. One of `"r.squared"`, `"slope"`, or `"p.value"`.

`point_alpha`
: Numeric. Alpha value for points.

`show_values`
: Logical. If `TRUE`, numeric values are printed inside heatmap tiles.

`temporal`
: Deprecated argument kept for compatibility with older examples. It is ignored.

Details

For `type = "regression_heatmap"`, pairwise simple linear regressions are fitted as `y ~ x` for every variable combination. The heatmap can display R^2, slope, or p-value.

Value

A `ggplot2` object, returned invisibly.

Examples

```
data(gf_nepa17)
data(ktm_clim_hourly)

zg <- phase.zg(df = gf_nepa17[1:800, ], TreeNum = 1)
zg_clim <- dm_add_climate(
  zg,
  ktm_clim_hourly,
  scale = "phase",
  mean_vars = c("temp", "VPD", "RH"),
  max_vars  = c("temp", "VPD"),
  sum_vars  = c("prec")
)

dm_plot_climate_compare(
  zg_clim,
  climate_vars = c("temp_mean_phase", "VPD_mean_phase", "RH_mean_phase"),
  scale = "cycle",
  type = "boxplot"
)

plot(
  zg_clim,
  climate_vars = c("temp_mean_phase", "VPD_mean_phase", "RH_mean_phase"),
  scale = "cycle",
  type = "boxplot"
)
```

dm_standardize *Standardize dendrometer series within seasonal years*

Description

Standardizes one or more dendrometer series within seasonal years so that multiple trees can be brought to a comparable scale while preserving their within-season temporal pattern.

Seasonal years are defined as:

- `"NH"`: Northern Hemisphere year, from **01 Jan** to **31 Dec**
- `"SH"`: Southern Hemisphere year, from **01 Jul** to **30 Jun** of the next year
- `"CS"`: Custom season defined by `CS_doys = c(doy1, doy2)`

For `season_type = "CS"`:

- if `CS_doys[1] <= CS_doys[2]`, the season stays within one calendar year
- if `CS_doys[1] > CS_doys[2]`, the season wraps across years

Standardization is applied separately for each dendrometer series and each seasonal year.

Usage

```
dm_standardize(
  df,
  season_type = c("NH", "SH", "CS"),
  CS_doys = NULL,
  method = c("center", "amplitude", "robust_amplitude", "minmax", "zscore",
    "robust_zscore", "percentile"),
  ref_type = c("first_value", "first_n_days", "ref_window"),
  ref_n_days = 7,
  ref_doys = NULL,
  q_low = 0.05,
  q_high = 0.95
)
```

Arguments

`df`	A data frame whose first column contains date-time (`yyyy-mm-dd HH:MM:SS`, `POSIXct`, or `Date`) and all remaining columns are dendrometer series.
`season_type`	One of `"NH"`, `"SH"`, or `"CS"`.
`CS_doys`	Optional numeric vector of length 2 defining the start and end DOY for `season_type = "CS"`.
`method`	Standardization method. One of: • `"center"`: subtract seasonal reference value • `"amplitude"`: subtract reference and divide by seasonal range

- "robust_amplitude": subtract reference and divide by Q_high - Q_low
- "minmax": rescale to (x - min)/(max - min)
- "zscore": seasonal z-score
- "robust_zscore": robust seasonal z-score using median and MAD
- "percentile": convert seasonal values to percentiles in $[0, 1]$

ref_type — Reference-value definition for methods that need a reference ("center", "amplitude", "robust_amplitude"). One of:

- "first_value": first non-missing value in the seasonal year
- "first_n_days": mean of the first ref_n_days unique days
- "ref_window": mean of values within ref_doys

ref_n_days — Number of initial days used when ref_type = "first_n_days".

ref_doys — Optional numeric vector of length 2 defining a DOY reference window when ref_type = "ref_window".

q_low — Lower quantile used by method = "robust_amplitude".

q_high — Upper quantile used by method = "robust_amplitude".

Value

A list of class "dm_standardized" containing:

- data: wide data frame with columns TIME, season_year, in_season, and standardized dendrometer series
- parameters: tibble with one row per tree and seasonal year, summarizing the reference value and scaling denominator used
- metadata: method and seasonal-year settings

For season_type = "CS", observations outside the custom season are retained in data, but their standardized values are set to NA and in_season = FALSE.

Examples

```
data(gf_nepa17)

# Northern Hemisphere seasonal-year standardization
out_nh <- dm_standardize(
  df = gf_nepa17,
  season_type = "NH",
  method = "robust_amplitude"
)

# Southern Hemisphere seasonal-year standardization
out_sh <- dm_standardize(
  df = gf_nepa17,
  season_type = "SH",
  method = "center"
)

# Custom season within one year
```

```
out_cs1 <- dm_standardize(
  df = gf_nepa17,
  season_type = "CS",
  CS_doys = c(100, 280),
  method = "robust_amplitude"
)

# Custom season wrapping across years
out_cs2 <- dm_standardize(
  df = gf_nepa17,
  season_type = "CS",
  CS_doys = c(250, 120),
  method = "percentile"
)

head(out_nh$data)
head(out_nh$parameters)
```

`dm_subdaily_clim` *Subdaily climate features for dendrometer analyses*

Description

Computes rolling-window and lagged climate features at subdaily resolution for direct linkage with point-level dendrometer outputs such as `ZG_phase` and `SC_phase`.

The input can be a standardized climate object returned by `read.climate()`, a raw data frame, or a valid file path accepted by `read.climate()`.

Usage

```
dm_subdaily_clim(
  clim_df,
  mean_vars = NULL,
  sum_vars = NULL,
  lag_vars = NULL,
  roll_hours = c(3, 6, 24),
  lag_hours = c(1, 3, 6, 24)
)
```

Arguments

`clim_df` Climate input. This can be:

- a standardized object returned by `read.climate()`
- a raw data frame with a time column in the first column or in a column named `TIME`
- a valid file path readable by `read.climate()`

mean_vars	Variables for rolling means.
sum_vars	Variables for rolling sums.
lag_vars	Variables for lagged features.
roll_hours	Numeric vector of rolling-window sizes in hours. Fractional values are allowed, e.g. 0.5 for 30 minutes.
lag_hours	Numeric vector of lag sizes in hours. Fractional values are allowed, e.g. 0.25 for 15 minutes.

Details

The function learns the temporal resolution automatically from the median time step in the TIME column. It works with hourly as well as minute-resolution data (for example 60-, 30-, 15-, 10-, or 5-minute data).

Rolling windows and lags are provided in hours and may be fractional:

- 0.25 = 15 minutes
- 0.5 = 30 minutes
- 1 = 1 hour
- 3 = 3 hours

If the user requests a rolling window or lag that is smaller than the inferred climate resolution, the function stops with an error.

If a requested window is not an exact multiple of the inferred resolution, it is rounded to the nearest number of time steps and a warning is issued.

Value

A tibble of class "subdaily_clim" with timestamp-level climate features added. The inferred temporal resolution in hours is stored in attr(x, "resolution_hours").

Examples

```
data(ktm_clim_hourly)

clim_sub <- dm_subdaily_clim(
  ktm_clim_hourly,
  mean_vars = c("temp", "VPD", "RH"),
  sum_vars  = c("prec"),
  lag_vars  = c("temp", "VPD", "RH"),
  roll_hours = c(1, 3, 6, 24),
  lag_hours  = c(1, 3, 6, 24)
)

head(clim_sub)
attr(clim_sub, "resolution_hours")
```

`dm_wavelet` *Wavelet analysis of dendrometer series*

Description

Performs continuous wavelet analysis on dendrometer time series.

The function can work with:

- raw dendrometer series from a data frame,
- detrended dendrometer series from `dm.detrend.fit()` output,
- first differences of either raw or detrended series.

It automatically:

- identifies the temporal resolution of the input series,
- regularizes the series to a complete time grid,
- optionally interpolates missing values,
- performs wavelet analysis separately for each selected tree/series,
- converts wavelet periods to hours for summary and plotting.

Usage

```
dm_wavelet(
  x,
  TreeNum = "all",
  source = c("auto", "raw", "detrended", "first_diff"),
  detrended_col = c("detrended_data"),
  na_action = c("interpolate", "fail"),
  loess_span = 0.75,
  dj = 1/20,
  lowerPeriod = NULL,
  upperPeriod = NULL,
  make_pval = TRUE,
  n_sim = 10,
  verbose = TRUE
)
```

Arguments

`x`	Input data. Either: **data.frame** First column must be time, remaining selected numeric columns are dendrometer series. **dm_detrended object** Uses `x$detrended_data` as input.
`TreeNum`	Either `"all"` to use all available dendrometer series, a numeric vector selecting series by position, or a character vector of series names.

source	Which series to analyze. One of: **‘"auto"‘** Uses raw series for a data frame input and detrended series for a dm_detrended object. **‘"raw"‘** Use raw dendrometer series from a data frame input. **‘"detrended"‘** Use detrended series from a dm_detrended object. **‘"first_diff"‘** Use first differences of the available input series.
detrended_col	For dm_detrended input, which table to use: "detrended_data".
na_action	How to handle missing values after completing the regular time grid. One of: **‘"interpolate"‘** Linearly interpolate internal gaps and carry ends forward/backward. **‘"fail"‘** Stop if missing values are found.
loess_span	Smoothing span passed to WaveletComp::analyze.wavelet().
dj	Frequency resolution parameter passed to WaveletComp::analyze.wavelet().
lowerPeriod	Optional lower period bound in native time units of the detected input resolution. If NULL, the default from WaveletComp::analyze.wavelet() is used.
upperPeriod	Optional upper period bound in native time units of the detected input resolution. If NULL, the default from WaveletComp::analyze.wavelet() is used.
make_pval	Logical; if TRUE, compute Monte Carlo significance.
n_sim	Number of simulations used when make_pval = TRUE.
verbose	Logical; if TRUE, prints a completion message.

Value

An object of class "dm_wavelet" with elements:

call The matched function call.

input_type Either "raw" or "dm_detrended".

source Series source actually used.

series Character vector of analyzed series names.

time_unit Detected time unit of the input series.

dt Detected time step in native units.

dt_hours Detected time step expressed in hours.

resolution_seconds Detected time step in seconds.

data_used Regularized time series used for analysis.

results Named list of wavelet results, one per series. Each entry contains the original WaveletComp object plus time and period-in-hours information.

settings Analysis settings.

`dm_wavelet_coherence` *Wavelet coherence between dendrometer and climate series*

Description

Computes cross-wavelet power and wavelet coherence between dendrometer series and climate variables using `WaveletComp::analyze.coherency()`.

The function:

- crops dendrometer and climate inputs to their common time window,
- detects their temporal resolutions,
- resamples both to the coarser detected resolution,
- optionally interpolates missing values,
- optionally uses first differences of the dendrometer series,
- computes pairwise coherence for all selected tree/climate combinations.

Usage

```
dm_wavelet_coherence(
  dm,
  clim,
  TreeNum = "all",
  clim_vars = "all",
  source = c("auto", "raw", "detrended", "first_diff"),
  detrended_col = c("detrended_data"),
  dm_fun = c("mean", "max", "min", "sum"),
  clim_funs = "mean",
  na_action = c("interpolate", "fail"),
  loess_span = 0,
  dj = 1/20,
  lowerPeriod = NULL,
  upperPeriod = NULL,
  window.type.t = 1,
  window.type.s = 1,
  window.size.t = 5,
  window.size.s = 1/4,
  make.pval = TRUE,
  make_pval = NULL,
  method = "white.noise",
  n.sim = 10,
  n_sim = NULL,
  verbose = TRUE
)
```

Arguments

dm	Dendrometer input. Either a data frame with time in the first column or an object of class "dm_detrended".
clim	Climate data frame. The first column must be time; remaining selected columns must be numeric climate variables.
TreeNum	Dendrometer series selector: "all", numeric indices, or character names.
clim_vars	Climate variable selector: "all", numeric indices, or character names.
source	One of "auto", "raw", "detrended", or "first_diff".
detrended_col	For dm_detrended input, which table to use.
dm_fun	Aggregation function for dendrometer resampling: one of "mean", "max", "min", "sum".
clim_funs	Either a single aggregation function applied to all climate variables, or a named character vector such as c(temp = "mean", VPD = "mean", prec = "sum").
na_action	One of "interpolate" or "fail".
loess_span	Detrending span passed to WaveletComp::analyze.coherency(). Use 0 to disable internal loess detrending.
dj	Frequency resolution passed to WaveletComp::analyze.coherency().
lowerPeriod	Optional lower period in native analysis units.
upperPeriod	Optional upper period in native analysis units.
window.type.t	Time-direction smoothing window type.
window.type.s	Scale-direction smoothing window type.
window.size.t	Time-direction smoothing window size.
window.size.s	Scale-direction smoothing window size.
make.pval	Logical. If TRUE, compute p-values.
make_pval	Optional alias for make.pval.
method	Surrogate generation method for p-values.
n.sim	Number of simulations.
n_sim	Optional alias for n.sim.
verbose	Logical.

Details

WaveletComp::analyze.coherency() expects two aligned series in one data frame plus a dt value in the analysis time units, and returns cross-wavelet power, coherence, phase angles, p-values, Fourier periods, and cone-of-influence fields.

Value

An object of class "dm_wavelet_coherence".

Examples

```
wc <- dm_wavelet_coherence(
  dm = gf_nepa17,
  clim = ktm_clim_hourly,
  TreeNum = 1,
  clim_vars = c("temp", "VPD"),
  source = "raw",
  dm_fun = "mean",
  clim_funs = c(temp = "mean", VPD = "mean"),
  loess_span = 0,
  make.pval = TRUE,
  n.sim = 10,
  verbose = FALSE
)

plot(wc)
plot(wc, type = "cross_power")
plot(wc, type = "average_coherence")
plot(wc, type = "phase")
plot(wc, type = "series")
```

---

dm_wavelet_reconstruct

Reconstruct or remove selected cycle components from a dm_wavelet object

---

Description

Reconstructs a selected oscillatory component, or a selected period band, from a `dm_wavelet` object using `WaveletComp::reconstruct()`.

Requested periods are supplied in **hours**. They are internally converted to the native wavelet period units used when `dm_wavelet()` was computed, then passed to `WaveletComp::reconstruct()`.

The function supports two modes:

- `mode = "extract"` returns the selected cycle or band itself.
- `mode = "remove"` returns the original series with the selected cycle or band removed.

Usage

```
dm_wavelet_reconstruct(
  x,
  series = NULL,
  mode = c("extract", "remove"),
  period_hours = NULL,
  lower_hours = NULL,
```

```
  upper_hours = NULL,
  lvl = 0,
  only_sig = TRUE,
  siglvl = 0.05,
  only_coi = FALSE,
  only_ridge = FALSE,
  rescale = FALSE,
  verbose = TRUE
)
```

Arguments

x	An object of class "dm_wavelet" returned by dm_wavelet().
series	Optional character vector of series names to reconstruct. If NULL, all available series are used.
mode	One of "extract" or "remove".
period_hours	Optional numeric vector of exact periods, in hours, to reconstruct. If supplied, lower_hours and upper_hours are ignored.
lower_hours	Optional lower period bound in hours for band reconstruction.
upper_hours	Optional upper period bound in hours for band reconstruction.
lvl	Minimum wavelet power level to include in the reconstruction.
only_sig	Logical. If TRUE, use wavelet power significance in reconstruction.
siglvl	Significance level used when only_sig = TRUE.
only_coi	Logical. If TRUE, restrict reconstruction to the cone of influence.
only_ridge	Logical. If TRUE, reconstruct from the power ridge only.
rescale	Logical. Passed to WaveletComp::reconstruct().
verbose	Logical. If TRUE, prints a completion message.

Value

An object of class "dm_wavelet_reconstruct" with elements:

call Matched function call.

series Selected series names.

mode Reconstruction mode, either "extract" or "remove".

selection A list describing the requested selection in hours and in native wavelet units.

results Named list of WaveletComp::reconstruct() outputs.

reconstructed_long Tidy table with columns TIME, series, original, reconstructed, difference, and filtered.

used_periods Per-series table of periods actually used in the reconstruction, in native units and in hours.

filtered_wide Wide table with TIME in the first column and one filtered series column per selected tree/series.

parent Minimal metadata copied from the parent dm_wavelet object.

Examples

```
wv <- dm_wavelet(
  x = gf_nepa17,
  TreeNum = 1:2,
  source = "raw",
  make_pval = TRUE,
  verbose = FALSE
)

# extract circadian component
rec_extract <- dm_wavelet_reconstruct(
  wv,
  mode = "extract",
  lower_hours = 20,
  upper_hours = 28,
  only_sig = TRUE
)

# remove circadian component
rec_remove <- dm_wavelet_reconstruct(
  wv,
  mode = "remove",
  lower_hours = 20,
  upper_hours = 28,
  only_sig = TRUE
)

head(rec_extract$filtered_wide)
head(rec_remove$filtered_wide)
```

---

gf_nepa17 *Dendrometer data of Kathmandu for 2017 with gap filled*

Description

The dendrometer data from three Chir pine tree collected in hourly resolution for 2017.

Usage

```
gf_nepa17
```

Format

A data frame with 8760 rows and 3 variables:

`Time` datetime time of data recording

`T2` double reading for first tree

`T3` double reading for second tree

`i.jump.locator` *Removing artefacts due to manual adjustments of dendrometers interactively*

Description

Dendrometers generally have limited memory capacity beyond which they stop recording. To keep the measurement ongoing, they should be adjusted periodically, which can cause positive or negative jumps in the data. This function locates these artefacts and interactively adjusts them one by one.

Usage

```
i.jump.locator(
  df,
  TreeNum,
  detection_method = c("manual", "auto"),
  manual_threshold = NULL,
  auto_method_penalty = 10,
  adjustment_method = c("window_median", "point_diff"),
  adjust_window = 5,
  plot_window = 20,
  auto_every_second = TRUE,
  set_jump_point_na = FALSE
)
```

Arguments

`df`	Data frame with first column containing date and time in the format `yyyy-mm-dd HH:MM:SS` and the dendrometer data in following columns.
`TreeNum`	Numerical value indicating the tree to be analysed. E.g. `1` refers to the first dendrometer data column in `df`.
`detection_method`	Either `"manual"` for threshold-based jump detection or `"auto"` for automatic detection using `changepoint::cpt.mean()` with `penalty = "Manual"`.
`manual_threshold`	Numeric threshold considered as artefact when `detection_method = "manual"`.
`auto_method_penalty`	Numeric manual penalty value used in `changepoint::cpt.mean()` when `detection_method = "auto"`. Larger values generally lead to fewer detected jumps.
`adjustment_method`	Either `"window_median"` (recommended) or `"point_diff"`. The former estimates the jump offset from local medians before and after the jump, while the latter uses the raw consecutive difference at the jump.
`adjust_window`	Integer window size used for robust jump-size estimation when `adjustment_method = "window_median"`.

`plot_window` Integer number of points shown before and after each jump in the interactive plot.

`auto_every_second`
Logical. If `TRUE`, keeps every second detected changepoint in automatic mode to mimic the legacy behavior.

`set_jump_point_na`
Logical. If `TRUE`, sets the exact jump point to `NA` after correction.

Value

A dataframe containing jump-free dendrometer data.

`jump.locator` *Removing artefacts due to manual adjustments of dendrometers automatically for more than one dendrometer*

Description

Dendrometers generally have limited memory capacity beyond which they stop recording. To keep the measurement ongoing, they should be adjusted periodically, which can cause positive or negative jumps in the data. This function locates these artefacts and adjusts them automatically. Unlike `i.jump.locator` , it can handle datasets with more than one dendrometer.

Usage

```
jump.locator(
  df,
  detection_method = c("manual", "auto"),
  manual_threshold = NULL,
  auto_method_penalty = 10,
  adjustment_method = c("window_median", "point_diff"),
  adjust_window = 5,
  auto_every_second = TRUE,
  set_jump_point_na = FALSE
)
```

Arguments

`df` Data frame with first column containing date and time in the format `yyyy-mm-dd HH:MM:SS` and the dendrometer data in following columns.

`detection_method`
Either `"manual"` for threshold-based jump detection or `"auto"` for automatic detection using `changepoint::cpt.mean()` with `penalty = "Manual"`.

`manual_threshold`
Numeric threshold considered as artefact when `detection_method = "manual"`.

`auto_method_penalty`
: Numeric manual penalty value used in `changepoint::cpt.mean()` when `detection_method = "auto"`. Larger values generally lead to fewer detected jumps.

`adjustment_method`
: Either `"window_median"` (recommended) or `"point_diff"`.

`adjust_window`
: Integer window size used for robust jump-size estimation when `adjustment_method = "window_median"`.

`auto_every_second`
: Logical. If `TRUE`, keeps every second detected changepoint in automatic mode to mimic the legacy behavior.

`set_jump_point_na`
: Logical. If `TRUE`, sets the exact jump point to `NA` after correction.

Value

A dataframe containing jump-free dendrometer data.

Examples

```
library(dendRoAnalyst)
data(nepa)

# Manual detection
jump_free_nepa <- jump.locator(
  df = nepa,
  detection_method = "manual",
  manual_threshold = 1
)

# Automatic detection with cpt.mean() and penalty = "Manual"
jump_free_nepa2 <- jump.locator(
  df = nepa,
  detection_method = "auto",
  auto_method_penalty = 10
)
```

---

ktm_clim_hourly *Hourly climate data for Kathmandu derived from ERA5-Land*

Description

Hourly near-surface climate data for Kathmandu, Nepal, extracted from the Copernicus Climate Change Service (C3S) ERA5-Land reanalysis. The dataset is included to demonstrate climate–dendrometer workflows in **dendRoAnalyst**, including joining hourly climate data with dendrometer observations, event-based analyses, and wavelet-based analyses.

Usage

```
ktm_clim_hourly
```

Format

A data frame with hourly observations and 5 variables:

`TIME` Date-time stamp of the hourly observation.

`temp` Air temperature in degree Celsius.

`prec` Precipitation in millimetres.

`VPD` Vapour pressure deficit in kPa.

`RH` Relative humidity in percent.

Details

`ktm_clim_hourly` contains a single-location hourly climate time series for Kathmandu derived from ERA5-Land. ERA5-Land is a global land-surface reanalysis produced by replaying the land component of ERA5 at enhanced spatial resolution and is widely used for land-surface and ecohydrological applications.

This dataset is intended as an example climate input for **dendRoAnalyst**. It can be used, for example, with functions that join dendrometer and climate data, calculate phase-climate relations, run event analyses, or perform wavelet and wavelet-coherence analyses.

Users should check the exact temporal coverage of the object in their local installation, for example with:

```
range(ktm_clim_hourly$TIME)
```

Source

Extracted from the Copernicus Climate Change Service (C3S) Climate Data Store ERA5-Land reanalysis product:

Muñoz-Sabater, J. (2019). ERA5-Land hourly data from 1950 to present. Copernicus Climate Change Service (C3S) Climate Data Store (CDS).

References

Muñoz-Sabater, J., Dutra, E., Agustí-Panareda, A., Albergel, C., Arduini, G., Balsamo, G., Boussetta, S., Choulga, M., Harrigan, S., Hersbach, H., Martens, B., Miralles, D. G., Piles, M., Rodríguez-Fernández, N. J., Zsoter, E., Buontempo, C., and Thépaut, J.-N. (2021). ERA5-Land: a state-of-the-art global reanalysis dataset for land applications. *Earth System Science Data*, 13, 4349–4383.

ktm_rain17 *Daily rainfall data of Kathmandu for 2017.*

Description

This file contains daily rainfall data of Kathmandu. The source of this data is 'Government of Nepal, Department of Hydrology and Meteorology'.

Usage

```
ktm_rain17
```

Format

A data frame with 365 rows and 2 variables:

`TIME` Date in YYYY-MM-DD format.

`rainfall` double rainfall in millimeters

Source

http://www.mfd.gov.np/city?id=31/

mean_detrended.dm *Calculate the mean detrended dendrometer series*

Description

Computes a mean detrended dendrometer series across multiple trees from the output of `dm.detrend.fit()`.

This is useful for creating one representative detrended series for a species, site, or treatment group after detrending individual dendrometer series.

Optionally, the function can:

- calculate a **robust mean** using a trimmed mean across trees,
- remove temporal autocorrelation from the mean detrended series using `forecast::auto.arima()`,
- rescale the autocorrelation-removed series so it stays non-negative and has mean = 1 within each vegetation season.

Usage

```
mean_detrended.dm(
  detrended_dm,
  series = NULL,
  ac1.remove = TRUE,
  robust.mean = TRUE,
  trim = 0.15,
  seasonal_rescale = TRUE
)
```

Arguments

detrended_dm	An object of class "dm_detrended" returned by dm.detrend.fit(), or a data frame in the same wide format as $detrended_data.
series	Optional character vector of tree/series names to include. Default is NULL, meaning all available detrended series are used.
ac1.remove	Logical. If TRUE, removes temporal autocorrelation from the mean detrended series using forecast::auto.arima() applied separately within each season. Default is TRUE.
robust.mean	Logical. If TRUE, calculates a trimmed mean across trees at each time step. Default is TRUE.
trim	Proportion to trim from each tail when robust.mean = TRUE. Default is 0.15.
seasonal_rescale	Logical. If TRUE, the autocorrelation-removed series is shifted to non-negative values and rescaled to mean = 1 within each season. Default is TRUE.

Value

A tibble of class "mean_dm_detrended" containing:

- metadata columns copied from the detrended input,
- STD_DDM: the mean detrended series,
- RES_DDM: the autocorrelation-removed mean detrended series (returned only when ac1.remove = TRUE).

Examples

```
fit1 <- dm.growth.fit(
  df = gf_nepa17,
  TreeNum = 1:2,
  method = "gompertz",
  year_mode = "yearly",
  verbose = FALSE
)

det1 <- dm.detrend.fit(fit1)

m_det <- mean_detrended.dm(det1)
head(m_det, 10)
```

`mov.cor.dm` *Running correlation between dendrometer data and climate*

Description

Calculates running correlations between a selected daily dendrometer summary and one or more climate variables. The user can select the daily dendrometer statistic, correlation method, optional bootstrap confidence intervals, and lagged / antecedent climate transformations.

Usage

```
mov.cor.dm(
  df,
  Clim,
  TreeNum,
  win_size,
  cor_method = c("pearson", "kendall", "spearman"),
  boot = FALSE,
  R = 1000,
  boot.ci = 0.05,
  set_seed = 1,
  dm_stat = c("mean", "min", "max", "median", "amplitude", "change"),
  clim_vars = NULL,
  lag_days = 0,
  accum_days = 1,
  clim_fun = "raw",
  min_complete = NULL,
  p_adjust_method = "BH"
)
```

Arguments

`df`	A data frame with the first column containing date-time in the format `yyyy-mm-dd HH:MM:SS` (or convertible to `POSIXct`) and dendrometer series in the following columns.
`Clim`	A data frame with the first column containing daily date (`yyyy-mm-dd`, `Date`, or convertible) and one or more climate variables in the following columns.
`TreeNum`	Integer indicating the dendrometer series to analyze.
`win_size`	Integer giving the running window size in days. Minimum is 18.
`cor_method`	Correlation method: one of `"pearson"`, `"kendall"`, or `"spearman"`.
`boot`	Logical. If `TRUE`, bootstrap confidence intervals are computed for each running correlation.
`R`	Integer number of bootstrap iterations.
`boot.ci`	Numeric confidence level selector: one of `0.01`, `0.05` (default), or `0.1`.
`set_seed`	Integer seed for reproducibility of bootstrap results.

dm_stat	Daily dendrometer statistic used for correlation. One of "mean", "min", "max", "median", "amplitude", or "change".
clim_vars	Optional character vector of climate variables to analyze. If NULL, all numeric climate variables are used.
lag_days	Climate lag in days. Can be: • one value for all selected climate variables • one value per selected climate variable • a named numeric vector keyed by climate variable name
accum_days	Antecedent window length in days for climate transformation. Can be scalar, per-variable, or a named numeric vector.
clim_fun	Climate transformation over the antecedent window. One of "raw", "mean", "sum", "max", "min", or "median". Can be scalar, per-variable, or a named character vector.
min_complete	Minimum number of complete paired observations required in a running window to calculate correlation. If NULL, defaults to max(3, ceiling(0.8 * win_size)).
p_adjust_method	Method for p-value adjustment in the non-bootstrap output. Passed to p.adjust(). Use "none" to skip adjustment.

Details

The dendrometer series is first aggregated to daily resolution. The daily dendrometer statistic used for correlation is controlled by dm_stat:

- "mean": daily mean dendrometer value
- "min": daily minimum
- "max": daily maximum
- "median": daily median
- "amplitude": daily amplitude (max - min)
- "change": day-to-day change in the daily mean

Users can choose the climate variables to analyze via clim_vars. Climate transformation settings can be given as:

- a single value applied to all selected climate variables
- an unnamed vector with one value per selected climate variable
- a named vector mapping each selected climate variable to its own setting

This applies to clim_fun, lag_days, and accum_days.

Value

A list with class "mov_cor_dm" (and "mov_cor_dm_boot" if bootstrapped) containing:

- results: named list of tibbles, one per climate variable
- metadata: analysis metadata
- call: the matched function call

Examples

```
library(dendRoAnalyst)
data(gf_nepa17)
data(ktm_rain17)

# one common climate transformation for all selected variables
out_corr <- mov.cor.dm(
  df = gf_nepa17,
  Clim = ktm_rain17,
  TreeNum = 1,
  win_size = 21,
  clim_fun = "raw"
)
print(out_corr)
summary(out_corr)

# variable-specific climate transformations
out_varfun <- mov.cor.dm(
  df = gf_nepa17,
  Clim = ktm_rain17,
  TreeNum = 1,
  win_size = 21,
  clim_vars = c("rainfall"),
  clim_fun = c(rainfall = "sum"),
  lag_days = c(rainfall = 1),
  accum_days = c(rainfall = 7)
)

# bootstrap confidence intervals
out_boot <- mov.cor.dm(
  df = gf_nepa17,
  Clim = ktm_rain17,
  TreeNum = 1,
  win_size = 21,
  boot = TRUE,
  R = 250
)
summary(out_boot)
```

---

nepa *Dendrometer data from Kathmandu*

Description

Dendrometer data from three Chir pine trees collected in hourly resolution for 2 years.

Usage

```
nepa
```

Format

A data frame with 14534 rows and 3 variables:

`Time` datetime time of data recording

`T2` double reading for first tree

`T3` double reading for second tree

---

nepa17 *Dendrometer data of Kathmandu for 2017*

Description

Dendrometer data from three Chir pine tree collected in hourly resolution for 2017.

Usage

```
nepa17
```

Format

A data frame with 8753 rows and 3 variables:

`Time` datetime time of data recording

`T2` double reading for first tree

`T3` double reading for second tree

---

nepa2 *Dendrometer data from Kathmandu version 2*

Description

Dendrometer data from three Chir pine trees collected in hourly resolution for 2 years with separated time.

Usage

```
nepa2
```

Format

A data frame with 14534 rows and 8 variables:

`year` numeric year of data recording

`month` numeric months of data recording

`day` numeric days of data recording

`hours` numeric hours of data recording

`minutes` numeric minutes of data recording

`seconds` numeric seconds of data recording

`T2` double reading for first tree

`T3` double reading for second tree

`network.interpolation` *Interpolate missing dendrometer values using a site network*

Description

Fills `NA` gaps in one or more focal dendrometer series by borrowing information from a reference network recorded at the same site and on the same time grid.

The focal dataset `df` is treated as the master time axis. The reference network is aligned to that axis, so missing timestamps already inserted into `df` are preserved during interpolation.

Two interpolation modes are available:

- `niMethod = "proportional"` uses the average relative change in the reference network between consecutive timestamps and propagates that change to the focal series.
- `niMethod = "linear"` fits a cross-sensor regression between reference values at time $t - 1$ and t, and predicts the focal value at time t from the focal value at time $t - 1$.

Optional bootstrap-based uncertainty limits can be attached, synthetic-gap validation can be used to assess recovery performance, and an optional post-processing step can detect and correct abrupt post-gap jumps in the interpolated series.

Usage

```
network.interpolation(
  df,
  referenceDF,
  niMethod = c("proportional", "linear"),
  n_boot = 1000,
  return_flags = TRUE,
  return_pi = TRUE,
  pi_for_all = FALSE,
  return_fit = FALSE,
  assess = FALSE,
```

```
  assess_lengths_steps = c(1, 2, 3, 6, 12, 24),
  assess_samples_per_length = 20,
  assess_buffer_steps = 1,
  assess_seed = NULL,
  assess_use_only_observed = TRUE,
  progress = interactive(),
  correct_gap_jumps = FALSE,
  jump_threshold = NULL,
  jump_adjustment_method = c("window_median", "point_diff"),
  jump_adjust_window = 5,
  jump_set_point_na = FALSE
)
```

Arguments

df	Data frame containing focal dendrometer data. The first column must be datetime (POSIXct, Date, or character convertible by lubridate::ymd_hms). Remaining columns are focal series.
referenceDF	Data frame containing reference dendrometer data. The first column must be datetime, and remaining columns are reference series.
niMethod	Character. Interpolation method. One of "proportional" or "linear".
n_boot	Integer. Number of bootstrap resamples used in proportional mode.
return_flags	Logical. If TRUE, add a logical column <series>_interp indicating which focal values were imputed.
return_pi	Logical. If TRUE, add 95% prediction-limit columns <series>_pi_lo and <series>_pi_hi.
pi_for_all	Logical. If TRUE, prediction limits are also attached for non-imputed rows where the model could be formed. If FALSE, limits are returned only for imputed rows.
return_fit	Logical. If TRUE, add <series>_fit columns containing the model-implied fitted value.
assess	Logical. If TRUE, run synthetic-gap validation.
assess_lengths_steps	Integer vector. Artificial gap lengths, expressed in number of rows/time steps, to test during validation.
assess_samples_per_length	Integer. Number of sampled artificial gaps per gap length and per focal series.
assess_buffer_steps	Integer. Number of observed steps required before and after each artificial gap when assess_use_only_observed = TRUE.
assess_seed	Integer or NULL. Random seed for reproducible validation sampling.
assess_use_only_observed	Logical. If TRUE, synthetic gaps are sampled only from windows fully observed in the focal series, including the requested buffer.
progress	Logical. If TRUE, display a single-line progress indicator.
correct_gap_jumps	Logical. If TRUE, inspect the first observed point after each imputed run and optionally remove a detected jump.

`jump_threshold` Numeric. Minimum absolute jump size required before a post-gap correction is applied. Required if `correct_gap_jumps = TRUE`.

`jump_adjustment_method`
Character. Method used to estimate the post-gap offset. One of `"window_median"` or `"point_diff"`.

`jump_adjust_window`
Integer. Window size used by `jump_adjustment_method = "window_median"`.

`jump_set_point_na`
Logical. If `TRUE`, the jump point itself is set to `NA` after correction.

Details

The function assumes that missing timestamps have already been inserted into `df` and corresponding measurements set to `NA`, for example with `dm.na.interpolation`.

For a focal series a, let a_{t-1} be the last available focal value before the current timestamp. Let b_{t-1} and b_t be the vectors of reference values at the previous and current timestamps.

In proportional mode, the relative reference change is

$$\delta = \text{mean}\left(\frac{b_t - b_{t-1}}{\max(b_{t-1}, \epsilon)}\right)$$

and the focal estimate is

$$\hat{a}_t = a_{t-1}(1 + \delta).$$

Bootstrap resampling of reference sensors is used to estimate a 95% interval.

In linear mode, a simple regression of b_t on b_{t-1} is fit across reference sensors at each step, and the focal value is predicted from a_{t-1} with a 95% prediction interval.

If fewer than two valid reference sensors are available at a timestamp pair, or if the focal previous value is missing, the focal value is left unchanged.

If `correct_gap_jumps = TRUE`, the function inspects the first observed point immediately after each imputed run. If a jump larger than `jump_threshold` is detected, the estimated offset is removed from that point onward. This is useful when a dendrometer resumes from a shifted baseline after a gap.

If `assess = TRUE`, the function inserts artificial gaps into observed sections of each focal series, interpolates them with the chosen method, and compares predictions with the true values. This produces both per-gap summaries and seasonal diagnostics of interpolation error.

The returned object is a data frame with class `"network_interpolation"`.

Value

A data frame with class `"network_interpolation"`.

The data frame contains the original `TIME` column, focal series columns, and optionally:

`<series>_interp` Logical flag for imputed rows.

`<series>_pi_lo`, `<series>_pi_hi` 95% prediction limits.

`<series>_fit` Model-implied fitted values.

`<series>_jump_correction` Numeric jump offset removed at the first observed point after an imputed run. `NA` means no correction was applied.

In addition, the following attributes are attached:

`network_original` Original focal data on the aligned time grid.

`network_reference` Reference data aligned to the focal time grid.

`network_diagnostics` Long-format diagnostics for plotting.

`network_summary` Per-series interpolation summary table.

`network_jump_corrections` Table of inspected and corrected post-gap jumps.

`network_validation_raw` Per-gap validation summary table, or `NULL`.

`network_validation_points` Point-level validation results, or `NULL`.

`network_validation_summary` Summarized validation metrics by series and gap length, or `NULL`.

`network_validation_seasonal` Seasonal validation diagnostics by day of year, or `NULL`.

`network_method` Interpolation method used.

`network_n_boot` Number of bootstrap resamples used.

`network_assessed` Logical indicating whether validation was run.

`network_jump_settings` List of jump-correction settings.

Interpretation

Smaller validation metrics such as `MAE`, `RMSE`, `MAPE`, and `MdAPE` indicate better gap recovery. `PI_coverage_95` indicates how well the nominal 95% uncertainty interval captures the true values under synthetic-gap validation. Seasonal diagnostics help identify periods of the year when network interpolation is more or less reliable.

If many post-gap corrections are applied or `max_abs_gap_jump` is large, it may indicate re-zeroing, sensor reset, or baseline discontinuities after gaps.

Notes

- The focal dataset `df` defines the output time grid.
- Reference series are aligned to that grid using the datetime column.
- At least two valid reference sensors are required at consecutive timestamps for interpolation.
- Jump correction is a post-processing step and does not alter the core interpolation logic.

See Also

`plot.network_interpolation`, `dm.na.interpolation`

Examples

```
#library(dendRoAnalyst)
#data("gf_nepa17")

## Create an artificial focal gap
#df1 <- gf_nepa17
#df1[40:50, "T2"] <- NA

## Build a small reference network
#ref <- cbind(gf_nepa17, gf_nepa17[, 2:3], gf_nepa17[, 2:3])
#colnames(ref) <- c("Time", "T1", "T2", "T3", "T4", "T5", "T6")

## Interpolate with uncertainty limits
#out <- network.interpolation(
#  df1, ref,
#  niMethod = "proportional",
#  n_boot = 100,
#  return_flags = TRUE,
#  return_pi = TRUE
#)

#head(out, 10)

# Interpolate with jump correction and validation
#out2 <- network.interpolation(
#  df1, ref,
#  niMethod = "proportional",
#  n_boot = 100,
#  return_flags = TRUE,
#  return_pi = TRUE,
#  assess = TRUE,
#  assess_lengths_steps = c(1, 2, 4),
#  correct_gap_jumps = TRUE,
#  jump_threshold = 0.05,
#  jump_adjustment_method = "window_median",
#  jump_adjust_window = 5
#)
# Plotting
#plot(out2)
#plot(out2, type = "compare")
#plot(out2, type = "seasonal_error")

# Extracting the information using attr()
#attr(out, "network_validation_summary")
#attr(out, "network_validation_points")
#attr(out, "network_validation_seasonal")
```

---

phase.sc *Application of the stem-cycle approach to classify dendrometer phases*

Description

Implements the **stem-cycle approach** (Downes et al., 1999; Deslauriers et al., 2011) to divide a dendrometer time series into three biologically meaningful phases:

1. **Shrinkage** (phase = 1): the dendrometer reading decreases compared to the previous reading.
2. **Expansion** (phase = 2): the dendrometer reading increases compared to the previous reading, but remains below the previous maximum.
3. **Increment** (phase = 3): the dendrometer reading exceeds the previous maximum (irreversible stem growth).

For each contiguous phase, the function calculates duration, magnitude, rate, and assigns day-of-year information. Optionally, the dendrometer series may be smoothed before phase calculation to reduce noise and spurious phase changes.

Usage

```
phase.sc(df, TreeNum, smoothing = NULL)
```

Arguments

`df`	A data frame with the first column containing date-time in the format `"yyyy-mm-dd HH:MM:SS"` (or convertible to `POSIXct`), followed by one or more dendrometer measurement columns (mm).
`TreeNum`	Integer. The index of the dendrometer column to analyze. For example, `TreeNum = 1` selects the first dendrometer series after the time column.
`smoothing`	Numeric or `NULL`. Length of the smoothing window in hours (1–24). If `NULL` (default), no smoothing is applied. If provided, the dendrometer series is smoothed using `smooth_dm(method = "median_mean")` prior to phase classification.

Details

Classification uses the cumulative maximum of the dendrometer series:

- If the cumulative maximum increases, the phase is labeled *Increment* (3).
- If the cumulative maximum is constant and the first difference is positive, the phase is *Expansion* (2).
- If the cumulative maximum is constant and the first difference is negative, the phase is *Shrinkage* (1).

The function returns both phase-level summaries (`SC_cycle`) and point-level labels (`SC_phase`). Optional smoothing uses `smooth_dm` with `method = "median_mean"` and a window length between 1–24 hours.

Value

A list of class `"SC_output"` containing:

SC_cycle A tibble with one row per contiguous phase, including:

- `Phases` – Phase type (1 = Shrinkage, 2 = Expansion, 3 = Increment)
- `Start`, `End` – `POSIXct` start and end time of the phase
- `Duration_h`, `Duration_m` – Phase duration (hours, minutes)
- `Magnitude` – Change in dendrometer value during the phase (measurement unit)
- `rate` – Rate of change expressed in (`Magnitude*1000/Duration_h`) (eg. μm/hour)
- `DOY` – Day-of-year at phase start

SC_phase A tibble of point-level values including:

- `TIME` – timestamp
- `dm` – dendrometer measurement
- `Phases` – phase assignment for each timestamp

References

Deslauriers A, Rossi S, Turcotte A, Morin H, Krause C (2011) A three-step procedure in SAS to analyze the time series from automatic dendrometers. *Dendrochronologia* 29:151–161. doi:10.1016/j.dendro.2011.01.008

Downes G, Beadle C, Worledge D (1999) Daily stem growth patterns in irrigated *Eucalyptus globulus* and *E. nitens* in relation to climate. *Trees* 14:102–111. doi:10.1007/PL00009752

See Also

`phase.zg` for the zero-growth approach; `smooth_dm` for smoothing dendrometer series.

Examples

```
library(dendRoAnalyst)
data(gf_nepa17)

# Apply stem-cycle approach without smoothing
sc1 <- phase.sc(df = gf_nepa17, TreeNum = 1)
head(sc1$SC_cycle, 5)
head(sc1$SC_phase, 5)

# Apply with 12-hour smoothing to reduce noise
sc2 <- phase.sc(df = gf_nepa17, TreeNum = 1, smoothing = 12)
head(sc2$SC_cycle, 5)
```

`phase.zg` *Apply the Zero-Growth (ZG) approach to segment dendrometer data into TWD/GRO phases*

Description

Implements the Zero-Growth approach (Zweifel et al., 2016) on a single dendrometer series to (i) classify each timestamp into **tree water deficit** (TWD; reversible shrinkage/expansion) or **growth** (GRO; irreversible expansion), (ii) summarize each contiguous phase (start/end, duration, magnitude, rates, TWD statistics), and (iii) optionally smooth the raw series prior to phase assignment to reduce spurious phase flips.

The growth line is computed as the running cumulative maximum of the (raw or smoothed) dendrometer series. TWD is defined as the vertical distance between the growth line and the observed measurement.

Usage

```
phase.zg(df, TreeNum, smoothing = NULL, beta = 0.1)
```

Arguments

`df`	A data frame with the **first column** containing date-time (character, `POSIXct`, or `Date` convertible to `POSIXct` using `lubridate::ymd_hms`) and the **subsequent columns** containing one or more dendrometer series (mm) at constant temporal resolution.
`TreeNum`	Integer index of the dendrometer series column to analyze. For example, `TreeNum = 1` refers to `df[[2]]` (the first dm column after the time column).
`smoothing`	`NULL` (default) for no smoothing, or a numeric value between 1 and 24 specifying the smoothing window in **hours** used by `smooth_dm(method = "median_mean")` for phase labeling.
`beta`	Numeric scalar giving the exponent applied to the effective loading duration in the ABr formula. Defaults to `0.1`. Set `beta = 0` to make ABr depend only on `max.twd`.

Details

Phase assignment. Let y_t be the dendrometer measurement (raw or smoothed). The growth line is $g_t = \max_{s \le t}(y_s)$. Points with $y_t = g_t$ are labeled **GRO** (phase = 2). Points with $y_t < g_t$ are labeled **TWD** (phase = 1). Contiguous runs of identical labels are summarized into individual phases.

Summaries per phase (ZG_cycle). For each phase, the function returns:

- `Phases` (1 = TWD, 2 = GRO)
- `Start`, `End` (`POSIXct`)
- `Duration_h` (hours)
- `Magnitude` (mm): for GRO only, computed from the *growth line* as $\Delta g = g_{\text{end}} - g_{\text{start}}$; `NA` for TWD
- `rate` (μm/h): `Magnitude * 1000 / Duration_h` (GRO only)
- `max.twd` (mm), `Max.twd.time` (`POSIXct`): peak TWD and its time (TWD only)
- `Avg.twd` (mm), `STD.twd` (mm): mean and SD of TWD within the phase (TWD only)

- AUC.load (mm*h): area under the TWD curve from phase start to Max.twd.time (TWD only)
- AUC.total (mm*h): area under the full TWD curve across the phase (TWD only)
- DOY: day-of-year of the phase start
- ABr.value: *Absolut Baumreaktion (ABr)* for TWD phases, defined here as

$$\mathrm{ABr} = \max(\mathrm{TWD}) \times \left(\frac{AUC_{load}}{\max(\mathrm{TWD})} \right)^{\beta}$$

  returned as NA for GRO phases. This metric anchors severity on the event peak while allowing the loading duration to contribute with exponent beta.

Point-level output (ZG_phase). The second element returns the input time series augmented with:

- Phases (1/2), TWD (= GRO - dm), and GRO (growth line).

Smoothing (optional). If smoothing is provided (hours), the function uses smooth_dm(..., method = "median_mean", window_hours = smoothing) to create a smoothed series used *only* for phase assignment; all magnitudes and TWD statistics are still computed against the *original* dm. This reduces short-lived phase flips due to noise. Valid range is 1–24 hours.

Temporal resolution. The function auto-detects the median sampling interval (in minutes) to parameterize smoothing. If multiple distinct intervals are detected, a warning is emitted (results may be degraded if resolution is inconsistent).

Notes & caveats.

- Negative Magnitude values in GRO phases indicate residual noise or insufficient smoothing; a warning is issued.
- Very short phases may occur around local extrema; consider smoothing or post-filtering minimal run lengths upstream if needed.
- Ensure timestamps are regular (or nearly regular) for best results.

Value

A named list of class "ZG_output" with two tibbles:

- ZG_cycle: one row per contiguous phase with columns Phases, Start, End, Duration_h, Magnitude, rate, max.twd, Max.twd.time, AUC.load, AUC.total, ABr.value, Avg.twd, STD.twd, DOY.
- ZG_phase: point-level data with columns TIME, dm, Phases, TWD, GRO.

Column definitions (ZG_cycle)

Phases Integer; 1 = TWD (reversible shrinkage/expansion), 2 = GRO (irreversible expansion).

Start, End POSIXct; phase boundaries.

Duration_h Numeric; phase duration in hours.

Magnitude Numeric; GRO-only millimeter change of growth line across the phase. NA for TWD.

rate Numeric; GRO-only rate in (Magnitude*1000/Duration_h).

max.twd Numeric; peak TWD within the TWD phase.

Max.twd.time `POSIXct`; time of `max.twd` within the TWD phase.

AUC.load Numeric; pre-peak area under the TWD curve in mm*h, integrated from phase start to `Max.twd.time`.

AUC.total Numeric; total area under the TWD curve in mm*h across the full TWD phase.

ABr.value Numeric; Absolut Baumreaktion value for TWD phases,

$$\max(\text{TWD}) \times \left(\frac{AUC_{load}}{\max(\text{TWD})} \right)^{\beta}$$

`NA` for GRO.

Avg.twd, STD.twd Numeric; mean and standard deviation of TWD in the phase.

DOY Integer; day-of-year for the phase start.

References

Zweifel R, Haeni M, Buchmann N, Eugster W (2016) Are trees able to grow in periods of stem shrinkage? *New Phytologist*, 211:839–849. doi:10.1111/nph.13995

Examples

```
library(dendRoAnalyst)
data(gf_nepa17)

# Minimal example (no smoothing)
zg <- phase.zg(df = gf_nepa17[1:600, ], TreeNum = 1)
head(zg$ZG_cycle, 5)
head(zg$ZG_phase, 5)

# With smoothing (e.g., 6 hours) to reduce short flips
zg6 <- phase.zg(df = gf_nepa17[1:600, ], TreeNum = 1, smoothing = 6, beta = 0.1)
subset(zg6$ZG_cycle, Phases == 1L)[1:5, c("Start","End","max.twd","ABr.value")]
```

---

plot.clim_twd_stats *Plot method for clim.twd.stats output*

Description

Plots grouped trajectories or grouped ID-level metrics from `clim.twd.stats()`.

Usage

```
## S3 method for class 'clim_twd_stats'
plot(
  x,
  y = NULL,
```

```
  type = c("trajectory", "id_metric"),
  ids = NULL,
  groups = NULL,
  band = c("none", "sd", "limit95", "both"),
 metric = c("lag_to_below_zero", "lag_to_max_adverse_decline", "lag_to_return_zero",
   "max_adverse_decline_value", "change_end_adverse", "change_end_full_period",
    "continuous_end_full_period"),
  facet_by = c("IDs", "group", "none"),
  legend_position = "bottom",
  main = NULL,
  ...
)
```

Arguments

x	An object of class "clim_twd_stats".
y	Unused.
type	One of "trajectory" or "id_metric".
ids	Optional numeric vector of IDs to plot.
groups	Optional character vector of group labels to plot.
band	One of "none", "sd", "limit95", or "both". Used for type = "trajectory".
metric	Metric column from x$id_summary for type = "id_metric".
facet_by	One of "IDs", "group", or "none".
legend_position	Legend position.
main	Optional title.
...	Further arguments passed to or from other methods.

Value

A ggplot2 object.

---

plot.clim_twd_test *Plot method for clim.twd.test output*

Description

Plots the data used in clim.twd.test() as grouped boxplots with jittered observations, faceted by stratum when applicable.

Usage

```
## S3 method for class 'clim_twd_test'
plot(
  x,
  y = NULL,
  show_points = TRUE,
  facet = TRUE,
  legend_position = "none",
  main = NULL,
  ...
)
```

Arguments

x	An object of class "clim_twd_test".
y	Unused.
show_points	Logical. If TRUE, add jittered points.
facet	Logical. If TRUE, facet by stratum when multiple strata exist.
legend_position	Legend position.
main	Optional plot title.
...	Further arguments passed to or from other methods.

Value

A ggplot2 object.

---

plot.daily_output *Plot method for daily dendrometer statistics*

Description

Unified S3 plotting method for objects returned by daily.data(). Missing grouping labels are removed before grouped boxplots are drawn so NA does not appear on the x-axis.

Usage

```
## S3 method for class 'daily_output'
plot(
  x,
  y = NULL,
  Year = NULL,
  DOY = NULL,
  type = c("summary", "amplitude", "minmax_ribbon", "timing", "timing_violin", "lag",
    "change", "boxplot", "heatmap"),
```

```
  stat = c("amplitude", "Max_diff", "mean", "median", "Max", "Min", "lag_h",
    "Time_min_h", "Time_max_h"),
  by = c("month", "month_of_year", "year"),
  box_style = c("boxplot", "violin", "both"),
 status_cols = c(growing = "forestgreen", shrinking = "firebrick", stable = "grey60"),
 timing_segment_cols = c(`max after min` = "grey65", `max before/equal min` = "black"),
  ...
)
```

Arguments

x	Object of class "daily_output".
y	Unused.
Year	Optional numeric year or vector of years for subsetting.
DOY	Optional numeric vector of length 2 giving the start and end day-of-year for subsetting.
type	Plot type. One of "summary", "amplitude", "minmax_ribbon", "timing", "timing_violin", "lag", "change", "boxplot", or "heatmap".
stat	Statistic used for type = "boxplot" or type = "heatmap". One of "amplitude", "Max_diff", "mean", "median", "Max", "Min", "lag_h", "Time_min_h", "Time_max_h".
by	Grouping for type = "boxplot": "month", "month_of_year", or "year". "month" uses a continuous calendar-date axis spanning the selected data window and supports multi-year plotting, for example 2022–2024. Empty calendar months may appear as gaps, but months with available data are not clipped at the start or end of the axis.
box_style	Style for type = "boxplot": "boxplot", "violin", or "both".
status_cols	Colors for Day_status.
timing_segment_cols	Named vector of two colors for the timing connector: one for "max after min" and one for "max before/equal min".
...	Unused.

Details

Plot types:

- "summary": daily Min, mean, median, and Max.
- "amplitude": daily amplitude through time.
- "minmax_ribbon": ribbon between daily Min and Max.
- "timing": per-day line connecting daily minimum and maximum timing.
- "timing_violin": violin distribution of Time_min_h and Time_max_h for the selected window.
- "lag": signed lag between daily maximum and minimum time.
- "change": day-to-day change in daily maximum Max_diff.

- "boxplot": grouped distributions of a chosen daily statistic. For by = "month", monthly groups are plotted on a continuous date axis. The axis limits are extended beyond the first and last plotted month so edge months are not clipped.
- "heatmap": year x day-of-year heatmap of a chosen daily statistic.

For type = "timing", the red and blue points are not connected across days. Instead, each day has its own segment connecting minimum and maximum time.

Value

A ggplot2 object, returned invisibly.

Examples

```
data(nepa17)
dd <- daily.data(df = nepa17[1:1000, ], TreeNum = 1)

plot(dd)
plot(dd, type = "timing")
plot(dd, type = "timing", Year = 2017, DOY = c(1, 6))
plot(dd, type = "timing_violin", Year = 2017, DOY = c(1, 6))
plot(dd, type = "boxplot", stat = "amplitude", by = "month")
plot(dd, type = "boxplot", stat = "amplitude", by = "month_of_year",
     box_style = "both")
plot(dd, type = "change")
```

---

plot.daily_output_clim

Plot climate-augmented daily dendrometer output

Description

S3 plotting method for objects of class daily_output_clim. This allows climate-augmented daily dendrometer outputs to be visualized directly with the generic plot() function.

Usage

```
## S3 method for class 'daily_output_clim'
plot(
  x,
  y = NULL,
  ...,
  climate_var = NULL,
  climate_vars = NULL,
  numeric_vars = NULL,
  compare = FALSE,
  temporal = NULL
)
```

Arguments

x	An object of class daily_output_clim.
y	Optional climate variable name passed as the second argument.
...	Additional arguments passed to dm_plot_climate() or dm_plot_climate_compare().
climate_var	Character. Name of one climate variable to plot.
climate_vars	Character vector of climate variables for comparison plots.
numeric_vars	Character vector of numeric variables used for correlation or regression heatmaps.
compare	Logical. If TRUE, dm_plot_climate_compare() is called.
temporal	Deprecated argument kept for compatibility with older examples. It is ignored.

Value

A ggplot2 object, returned invisibly.

Examples

```
# plot(daily_clim_output, climate_var = "temp_mean", type = "timeseries")
```

---

plot.dm_detrended *Plot detrended dendrometer series*

Description

S3 plotting method for objects returned by [dm.detrend.fit()].

The default plot compares:

- original daily dendrometer series,
- fitted curve reconstructed on the original daily scale,
- residuals (‘observed - fitted_original‘),
- detrended standardized series.

Usage

```
## S3 method for class 'dm_detrended'
plot(
  x,
  y = NULL,
  type = c("compare", "fit", "residual", "detrended", "boxplot"),
  series = NULL,
  seasons = NULL,
  x_axis = c("default", "date", "season_day", "doy"),
  facet_by = c("series", "season", "none"),
```

```
  ncol = NULL,
  box_group = c("series", "season"),
  show_observed = TRUE,
  show_fitted = TRUE,
  point_alpha = 0.7,
  line_width = 0.8,
  legend_position = "right",
  ...
)
```

Arguments

x	An object of class "dm_detrended" returned by [dm.detrend.fit()].
y	Unused.
type	Plot type. One of: **‘"compare"‘** Default three-panel comparison of original daily vs fitted, residuals, and detrended standardized series. **‘"fit"‘** Original daily and fitted original-scale values only. **‘"residual"‘** Residuals only. **‘"detrended"‘** Detrended standardized series only. **‘"boxplot"‘** Distribution of detrended standardized values by series or season.
series	Optional character vector of dendrometer series to plot. Default is NULL, meaning all available series are used.
seasons	Optional character vector of vegetation-season labels to plot. Default is NULL, meaning all seasons are used.
x_axis	Character string controlling the x-axis. One of: **‘"default"‘** Uses date for ‘"compare"‘ and ‘"fit"‘, and season day for ‘"residual"‘ and ‘"detrended"‘. **‘"date"‘** Use actual calendar date. **‘"season_day"‘** Use vegetation-season day. **‘"doy"‘** Use calendar day-of-year.
facet_by	Character string controlling faceting. One of: **‘"series"‘** Facet by dendrometer series. **‘"season"‘** Facet by vegetation season. **‘"none"‘** No faceting.
ncol	Optional integer giving the number of columns in faceted plots where [ggplot2::facet_wrap()] is used.
box_group	For type = "boxplot", grouping variable on the x-axis. One of "series" or "season".
show_observed	Logical. If TRUE, original daily observations are shown in plot types where relevant. Default is TRUE.
show_fitted	Logical. If TRUE, fitted original-scale values are shown in plot types where relevant. Default is TRUE.

point_alpha	Numeric alpha level used for observed points. Default is 0.7.
line_width	Numeric line width used for fitted, residual, and detrended lines. Default is 0.8.
legend_position	Character string specifying legend position. Default is "right".
...	Further arguments passed to or from other methods.

Value

A ggplot2 object.

Examples

```
fit1 <- dm.growth.fit(
  df = gf_nepa17,
  TreeNum = 1:2,
  method = "gompertz",
  year_mode = "yearly",
  verbose = FALSE
)

det1 <- dm.detrend.fit(fit1)

plot(det1)
plot(det1, type = "fit")
plot(det1, type = "residual")
plot(det1, type = "detrended")
plot(det1, type = "boxplot")
plot(det1, type = "compare", facet_by = "series")
plot(det1, type = "compare", facet_by = "season")
```

---

plot.dm_epoch *Plot a dm_epoch object*

Description

Plot a dm_epoch object

Usage

```
## S3 method for class 'dm_epoch'
plot(
  x,
  y = NULL,
  type = c("composite", "heatmap", "difference"),
  variables = NULL,
  facet = TRUE,
```

```
  show_ci = TRUE,
  legend_position = "right",
  ...
)
```

Arguments

x	Object of class "dm_epoch".
y	Unused.
type	Plot type. One of "composite", "heatmap", or "difference".
variables	Optional climate variables to plot.
facet	Logical. If TRUE, facet by variable.
show_ci	Logical. If TRUE and x is a "dm_epoch_test" object, show the null confidence ribbon.
legend_position	Legend position passed to [ggplot2::theme()].
...	Unused.

Value

A ggplot object.

---

plot.dm_growth_evaluation

Plot growth-fitting evaluation statistics

Description

Creates ggplot2-based comparison plots for objects returned by [dm.growth.evaluate()].

Usage

```
## S3 method for class 'dm_growth_evaluation'
plot(
  x,
 metric = c("rmse", "mae", "bias", "abs_bias", "r2", "correlation", "nrmse", "rss",
    "aic_approx", "bic_approx"),
  type = c("boxplot", "mean", "heatmap"),
  order_methods = TRUE,
  decreasing = NULL,
  show_points = TRUE,
  show_errorbar = TRUE,
  heatmap_label = c("series_fit", "series", "fit_id"),
  na.rm = TRUE,
  ...
)
```

Arguments

x	An object of class "dm_growth_evaluation".
metric	Evaluation metric to plot. One of "rmse", "mae", "bias", "abs_bias", "r2", "correlation", "nrmse", "rss", "aic_approx", or "bic_approx".
type	Plot type. One of "boxplot", "mean", or "heatmap".
order_methods	Logical. If TRUE, methods are ordered by their average value for the chosen metric.
decreasing	Logical or NULL. Controls method ordering. If NULL, metrics where smaller values are better are ordered ascending, and metrics where larger values are better are ordered descending.
show_points	Logical. If TRUE, add individual fit points to "boxplot" and "mean" displays.
show_errorbar	Logical. If TRUE, show standard-error bars when type = "mean".
heatmap_label	Character string used to label rows in the heatmap. One of "series_fit", "series", or "fit_id".
na.rm	Logical. If TRUE, remove missing metric values before plotting.
...	Further arguments passed to plotting methods.

Value

A ggplot object.

See Also

[dm.growth.evaluate()]

---

plot.dm_growth_fit *Plot dendrometer growth-fit results*

Description

Creates **ggplot2**-based plots for objects returned by [dm.growth.fit()] and [dm.growth.fit.double()].

The plotting method supports multiple views of fitted dendrometer growth curves, including observed versus fitted trajectories, residuals, timing summaries, overlays of multiple curves, and distributions of fitted model parameters.

The x-axis can be displayed as vegetation-season day, calendar day-of-year (DOY), or actual date, depending on the selected x_axis argument and the type of plot.

When x_axis = "date" and faceting separates individual years or series-year combinations, each facet receives its own x-axis range. This avoids plotting one yearly fitted curve inside the full dendrometer time window when several seasons are present in the input object.

Usage

```
## S3 method for class 'dm_growth_fit'
plot(
  x,
  type = c("fit", "season", "residuals", "timing", "overlay", "parameters"),
  series = NULL,
  fit_id = NULL,
  facet_by = c("default", "tree", "year", "none"),
  ncol = NULL,
  normalize = FALSE,
  x_axis = c("default", "season_day", "doy", "date"),
  observed_source = c("processed", "original_daily"),
  show_observed = TRUE,
  show_fitted = TRUE,
  show_timing = TRUE,
  point_alpha = 0.7,
  line_width = 0.8,
  legend_position = "right",
  ...
)
```

Arguments

x	An object of class "dm_growth_fit" returned by [dm.growth.fit()] or [dm.growth.fit.double()].
type	Character string specifying the plot type. One of "fit", "season", "residuals", "timing", "overlay", or "parameters".
series	Optional filter selecting one or more dendrometer series to plot. May be a character vector of series names. Default is NULL, meaning all available series are used.
fit_id	Optional filter selecting one or more fit identifiers to plot. May be a character or numeric vector. Numeric values are coerced internally to character. Default is NULL, meaning all fits are used.
facet_by	Character string controlling faceting layout. One of "default", "tree", "year", or "none".
ncol	Optional integer giving the number of columns in faceted plots. Passed to [ggplot2::facet_wrap()].
normalize	Logical. If TRUE, observed and fitted values are divided by the maximum observed or fitted value within each curve.
x_axis	Character string controlling the x-axis representation. One of "default", "season_day", "doy", or "date".
observed_source	Character string controlling which observed daily series is plotted against the fitted curve. One of "processed" or "original_daily".
show_observed	Logical. If TRUE, observed values are shown in plot types where observed data are relevant.

show_fitted	Logical. If TRUE, fitted values are shown in plot types where fitted curves are relevant.
show_timing	Logical. If TRUE, timing markers are added to "fit" and "season" plots whenever timing information is available in x$fit_statistics.
point_alpha	Numeric alpha level used for observed points.
line_width	Numeric line width used for fitted curves.
legend_position	Character string specifying legend position, passed to [ggplot2::theme()].
...	Further arguments passed to or from other methods.

Details

The plotting method returns a ggplot object. The returned plot can be further modified using normal **ggplot2** syntax.

Timing markers are taken from the fit_statistics table inside the "dm_growth_fit" object:

- growth_start_* and growth_end_* represent growing-season timing based on cumulative fitted growth.
- rate_start_* and rate_end_* represent active-growth timing based on the fitted growth-rate curve.
- For double-growth fits, pulse-specific timing such as pulse1_start_*, pulse2_start_*, and separator_* is used in type = "timing" when available.

For yearly fits in southern hemisphere or cross-year custom seasons, calendar timing variables such as growth_start_day and rate_start_day are interpreted as true calendar DOY, while *_season_day variables represent day counts relative to vegetation season start.

In pooled fits, date-based and calendar-DOY timing fields may be unavailable, because pooled fits are not anchored to a single season start date.

Value

A ggplot object.

See Also

[dm.growth.fit()], [dm.growth.fit.double()], [summary.dm_growth_fit()]

Examples

```
# fit <- dm.growth.fit(...)
# plot(fit, type = "fit")
# plot(fit, type = "fit", facet_by = "year", x_axis = "date")
# plot(fit, type = "fit", facet_by = "year", x_axis = "season_day")
# plot(fit, type = "residuals", facet_by = "year")
# plot(fit, type = "timing")
# plot(fit, type = "parameters")
```

plot.dm_na_interpolation *Plot method for dendrometer NA interpolation results*

Description

S3 plot method for objects returned by dm.na.interpolation.

Usage

```
## S3 method for class 'dm_na_interpolation'
plot(
  x,
  type = NULL,
  series = NULL,
  metric = "MdAPE",
  original = NULL,
  start = NULL,
  end = NULL,
  free_y = TRUE,
  ncol = 1,
  facet = TRUE,
  empty_gaps = c("plot", "error"),
  ...
)
```

Arguments

x	Object returned by dm.na.interpolation().
type	Character. Plot type: "gaps", "interpolation", or "assessment". If omitted, the default is: • "assessment" when assessment exists, • "interpolation" when filled data exist, • "gaps" when gaps exist but data were not filled, • otherwise "interpolation".
series	Optional character vector of series names to plot.
metric	Character assessment metric used when type = "assessment".
original	Optional original input data frame used before interpolation.
start	Optional start datetime for subsetting.
end	Optional end datetime for subsetting.
free_y	Logical. If TRUE, facets use free y scales.
ncol	Integer. Number of facet columns.
facet	Logical. If TRUE, facet assessment plots by series.

empty_gaps	Character. Behavior when type = "gaps" but no gaps are present: "plot" for an informative empty plot, or "error".
...	Further arguments.

Value

A ggplot2 object.

Examples

```
library(dendRoAnalyst)
data(nepa17)

## No gaps: defaults to time-series plot
#res0 <- dm.na.interpolation(nepa17[1:1000, ], resolution = 60)
#plot(res0)
#plot(res0, type = "gaps")
#plot(res0, type = "gaps", empty_gaps = "error")

## Filled data
#res1 <- dm.na.interpolation(
#  nepa17[1:1000, ],
#  resolution = 60,
#  fill = TRUE,
#  method = "spline"
#)
#plot(res1)
#plot(res1, type = "gaps")
#plot(res1, type = "interpolation", original = nepa17[1:1000, ])

## Assessed data
#res2 <- dm.na.interpolation(
#  nepa17[1:1000, ],
#  resolution = 60,
#  fill = TRUE,
#  method = "seasonal",
#  assess = TRUE
#)
#plot(res2)
#plot(res2, type = "assessment", metric = "MdAPE")
```

---

plot.dm_standardized *Plot method for standardized dendrometer output*

Description

S3 plotting method for objects returned by dm_standardize().

Usage

```
## S3 method for class 'dm_standardized'
plot(
  x,
  y = NULL,
  trees = "all",
  type = c("series", "seasonal", "heatmap", "boxplot"),
  x_axis = c("time", "doy", "season_doy"),
  facet_by = c("tree", "season_year", "none"),
  legend_by = c("season_year", "tree", "none"),
  in_season_only = TRUE,
  box_group = c("tree", "season_year"),
  alpha = 0.8,
  line_size = 0.45,
  point_size = 1.6,
  ...
)
```

Arguments

x	Object of class "dm_standardized" returned by dm_standardize().
y	Unused.
trees	Character vector of dendrometer series to plot, or "all" for all series.
type	Plot type. One of: • "series": standardized time series over calendar time • "seasonal": standardized seasonal trajectories by seasonal day • "heatmap": heatmap of standardized values by seasonal day and seasonal year • "boxplot": distribution of standardized values
x_axis	For type = "series", x-axis style: "time", "doy", or "season_doy".
facet_by	Faceting option. One of "tree", "season_year", or "none".
legend_by	Legend grouping. One of "tree", "season_year", or "none".
in_season_only	Logical. If TRUE, only observations inside the defined season are plotted.
box_group	For type = "boxplot", grouping variable on the x-axis: "tree" or "season_year".
alpha	Line or point transparency.
line_size	Line width.
point_size	Point size.
...	Unused.

Value

A ggplot2 object, returned invisibly.

Examples

```
data(gf_nepa17)

out_std <- dm_standardize(
  df = gf_nepa17,
  season_type = "NH",
  method = "robust_amplitude"
)

plot(out_std)
plot(out_std, type = "seasonal")
plot(out_std, type = "seasonal", facet_by = "tree", legend_by = "season_year")
plot(out_std, type = "seasonal", facet_by = "season_year", legend_by = "tree")
plot(out_std, type = "heatmap", facet_by = "tree")
plot(out_std, type = "boxplot", facet_by = "tree", legend_by = "season_year")
```

---

plot.dm_wavelet *Plot method for wavelet analysis output*

Description

S3 plotting method for objects returned by `dm_wavelet()`.

The function provides three main visualization types:

- `"series"`: plots the analyzed input dendrometer series over time,
- `"average"`: plots the average wavelet power spectrum against period,
- `"power"`: plots the full wavelet power spectrum as a time-period image.

Periods are shown in **hours**, regardless of the original temporal resolution of the input series. For power plots, the x-axis shows the actual time series timestamps and the y-axis shows the wavelet period in hours.

Usage

```
## S3 method for class 'dm_wavelet'
plot(
  x,
  y = NULL,
  series = NULL,
  type = c("power", "average", "series"),
  facet = TRUE,
  log_period = TRUE,
  log_power = TRUE,
  clip_quantile = c(0.01, 0.99),
  show_sig = TRUE,
```

```
  show_coi = TRUE,
  coi_fill = "white",
  coi_alpha = 0.45,
  siglvl = 0.05,
  sig_color = "black",
  sig_size = 0.4,
  main = NULL,
  ...
)
```

Arguments

x	An object of class "dm_wavelet" returned by dm_wavelet().
y	Unused.
series	Optional character vector giving one or more series names to plot. If NULL, the first available series is used.
type	Character string specifying the plot type. One of: **‘"power"‘** Wavelet power spectrum as a raster image. **‘"average"‘** Average wavelet power spectrum across the full time series. **‘"series"‘** Original analyzed series over time.
facet	Logical. If TRUE and more than one series is selected, panels are faceted by series.
log_period	Logical. If TRUE, the period axis is shown on a log10 scale where applicable.
log_power	Logical. If TRUE and type = "power", the plotted wavelet power is transformed using log10(power + eps) to improve contrast.
clip_quantile	Optional numeric vector of length 2 giving lower and upper quantiles used to clip wavelet power before plotting, for example c(0.01, 0.99). This is useful when a few extreme values dominate the colour scale. Use NULL to disable clipping.
show_sig	Logical. If TRUE, overlays significance information when available. For type = "average", significant periods are marked as points. For type = "power", significant regions are shown as contour lines.
show_coi	Logical. If TRUE and type = "power", the cone of influence (COI) is added as a shaded overlay.
coi_fill	Fill colour used for the cone of influence shading.
coi_alpha	Numeric transparency of the cone of influence shading.
siglvl	Numeric significance threshold between 0 and 1. Default is 0.05.
sig_color	Colour used for significance overlays.
sig_size	Line width or point size used for significance overlays.
main	Optional plot title. If NULL, a default title is used.
...	Further arguments passed to or from other methods. Currently unused.

Details

For `type = "power"`, the function uses the wavelet power matrix stored in the `"dm_wavelet"` object and converts the Fourier periods to hours. The power plot may also show:

- significance contours, when p-values are available,
- the cone of influence (COI), when COI information is available.

For `type = "average"`, the function plots the average wavelet power spectrum and may optionally indicate significant periods when average-spectrum p-values are available.

For `type = "series"`, the original analyzed series are plotted without any wavelet transformation.

Value

A `ggplot2` object.

See Also

`dm_wavelet`, `ggplot`

Examples

```
wv <- dm_wavelet(
  x = gf_nepa17,
  TreeNum = 1:2,
  source = "raw",
  make_pval = TRUE,
  verbose = FALSE
)

# original series
plot(wv, type = "series")

# average wavelet power
plot(wv, type = "average")

# full power spectrum
plot(wv, type = "power")

# one selected series
plot(wv, series = names(wv$results)[1], type = "power")

# stronger contrast in the power plot
plot(
  wv,
  type = "power",
  log_power = TRUE,
  clip_quantile = c(0.05, 0.95)
)
```

plot.dm_wavelet_coherence *Plot method for dm_wavelet_coherence objects*

Description

Plot method for dm_wavelet_coherence objects

Usage

```
## S3 method for class 'dm_wavelet_coherence'
plot(
  x,
  y = NULL,
  pair = NULL,
  type = c("coherence", "cross_power", "average_coherence", "average_cross_power",
    "phase", "series"),
  facet = TRUE,
  log_period = TRUE,
  log_power = TRUE,
  clip_quantile = c(0.01, 0.99),
  show_sig = TRUE,
  show_coi = TRUE,
  siglvl = 0.05,
  sig_color = "black",
  sig_size = 0.4,
  coi_fill = "white",
  coi_alpha = 0.45,
  main = NULL,
  ...
)
```

Arguments

x	An object of class "dm_wavelet_coherence".
y	Unused.
pair	Optional pair name(s) to plot. If NULL, the first pair is used.
type	One of "coherence", "cross_power", "average_coherence", "average_cross_power", "phase", or "series".
facet	Logical. If TRUE and multiple pairs are selected, facet them.
log_period	Logical.
log_power	Logical for raster intensity.
clip_quantile	Optional clipping quantiles for raster intensity.
show_sig	Logical.

show_coi	Logical.
siglvl	Significance level.
sig_color	Significance contour color.
sig_size	Significance contour linewidth.
coi_fill	COI fill color.
coi_alpha	COI alpha.
main	Optional title.
...	Unused.

Value

A ggplot2 object.

`plot.dm_wavelet_reconstruct` *Plot method for dm_wavelet_reconstruct objects*

Description

Plots reconstructed or filtered cycle components extracted by `dm_wavelet_reconstruct()`.

Usage

```
## S3 method for class 'dm_wavelet_reconstruct'
plot(
  x,
  y = NULL,
  series = NULL,
  type = c("compare", "reconstructed", "difference", "filtered"),
  facet = TRUE,
  legend_position = "right",
  line_width = 0.8,
  alpha = 0.7,
  main = NULL,
  ...
)
```

Arguments

x	An object of class "dm_wavelet_reconstruct".
y	Unused.
series	Optional character vector of series names to plot. If NULL, all available reconstructed series are used.
type	One of:

	"compare" Original and reconstructed series together.
	"reconstructed" Reconstructed component only.
	"difference" Original minus reconstructed.
	"filtered" Directly plot the returned filtered series. This is the extracted component for `mode = "extract"` and the component-removed series for `mode = "remove"`.
`facet`	Logical. If `TRUE`, facet by series.
`legend_position`	Legend position passed to ggplot2.
`line_width`	Line width.
`alpha`	Alpha transparency for original series in compare plots.
`main`	Optional title.
`...`	Further arguments passed to or from other methods.

Value

A `ggplot2` object.

`plot.mean_dm_detrended`

Plot mean detrended dendrometer series

Description

S3 plotting method for objects returned by `mean_detrended.dm()`.

It can plot:

- the mean detrended series (`STD_DDM`),
- the autocorrelation-removed mean detrended series (`RES_DDM`),
- or both together.

Usage

```
## S3 method for class 'mean_dm_detrended'
plot(
  x,
  y = NULL,
  type = c("series", "seasonal", "boxplot"),
  value = c("both", "STD_DDM", "RES_DDM"),
  seasons = NULL,
  x_axis = c("default", "date", "doy", "season_day"),
  facet_by = c("none", "season", "metric"),
  ncol = NULL,
  box_group = c("metric", "season"),
```

```
  alpha = 0.8,
  line_width = 0.8,
  point_size = 1.4,
  legend_position = "right",
  ...
)
```

Arguments

x	An object of class "mean_dm_detrended" returned by mean_detrended.dm().
y	Unused.
type	Plot type. One of: **‘"series"‘** Plot the time series over calendar time, DOY, or season day. **‘"seasonal"‘** Overlay seasons on a season-day scale. **‘"boxplot"‘** Show boxplots of the detrended values by season or metric.
value	Which variable to plot. One of: **‘"both"‘** Plot both STD_DDM and RES_DDM when available. **‘"STD_DDM"‘** Plot only the mean detrended series. **‘"RES_DDM"‘** Plot only the autocorrelation-removed mean detrended series.
seasons	Optional character vector of season_label values to retain.
x_axis	Character string controlling the x-axis. One of: **‘"default"‘** Uses calendar date for type = "series" and season day for type = "seasonal". **‘"date"‘** Use actual calendar date. **‘"doy"‘** Use calendar day-of-year. **‘"season_day"‘** Use vegetation season day.
facet_by	Character string controlling faceting. One of: **‘"none"‘** No faceting. **‘"season"‘** Facet by season_label. **‘"metric"‘** Facet by metric (STD_DDM, RES_DDM).
ncol	Optional integer giving the number of facet columns.
box_group	For type = "boxplot", grouping variable on the x-axis. One of "metric" or "season".
alpha	Numeric alpha transparency for lines/points.
line_width	Numeric line width.
point_size	Numeric point size.
legend_position	Character legend position passed to ggplot2.
...	Further arguments passed to or from other methods.

Value

A ggplot2 object.

plot.mov_cor_dm *Plot method for moving dendrometer-climate correlation*

Description

S3 plotting method for output of `mov.cor.dm()`. Supports heatmaps, line plots, faceted line plots, peak-correlation summaries, and comparison across multiple `mov.cor.dm()` objects.

Usage

```
## S3 method for class 'mov_cor_dm'
plot(
  x,
  y = NULL,
  sig.only = TRUE,
  ci = 0.95,
  clim_vars = "all",
  type = c("heatmap", "line", "facet", "peak", "compare"),
  x_axis = c("time", "doy"),
  use_adjusted = TRUE,
  show_na = TRUE,
  show_ci = FALSE,
  sig_mode = c("outline", "point", "filter", "none"),
  annotate_peak = FALSE,
  show_window_label = TRUE,
  compare_with = NULL,
  compare_labels = NULL,
  low_col = "red",
  mid_col = "white",
  high_col = "blue",
  na_col = "grey79",
  line_size = 0.5,
  point_size = 1.8,
  alpha_sig = 0.9,
  ...
)
```

Arguments

`x`	Object returned by `mov.cor.dm()`.
`y`	Unused.
`sig.only`	Logical. If `TRUE`, only significant windows are shown in heatmaps and significance is highlighted in other plot types.
`ci`	Numeric confidence level between 0 and 1. For non-bootstrap results, significance is based on `p_val < 1 - ci` or `p_adj < 1 - ci`. For bootstrap results, the stored logical `significance` column is used.

clim_vars	Character vector of climate variables to plot, or "all" for all variables.
type	Plot type. One of "heatmap", "line", "facet", "peak", or "compare".
x_axis	X-axis style. One of "time" or "doy".
use_adjusted	Logical. For non-bootstrap objects, if TRUE, significance uses p_adj; otherwise p_val.
show_na	Logical. If TRUE, missing values are shown in heatmaps.
show_ci	Logical. If TRUE, bootstrap confidence intervals are shown in line/facet/compare plots when available.
sig_mode	One of "outline", "point", "filter", or "none" describing how significance is displayed in non-heatmap plots.
annotate_peak	Logical. If TRUE, the strongest absolute correlation is marked for each climate variable in line/facet/compare plots.
show_window_label	Logical. If TRUE, type = "peak" adds the climate transformation settings above each bar.
compare_with	Optional list of additional mov_cor_dm objects for comparison. Used only when type = "compare".
compare_labels	Optional character vector of labels for the comparison objects. If NULL, defaults are generated automatically.
low_col	Colour for negative correlations.
mid_col	Colour for zero correlations.
high_col	Colour for positive correlations.
na_col	Fill colour for missing values in heatmaps.
line_size	Numeric line width for line plots.
point_size	Numeric point size for significance markers and peak markers.
alpha_sig	Numeric transparency used for significance highlighting.
...	Unused.

Value

A ggplot2 object, returned invisibly.

`plot.network_interpolation`

Plot method for network interpolation output

Description

S3 plot method for objects returned by network.interpolation.

Depending on type, the method can display:

- interpolated series through time,
- uncertainty width,
- reference-network availability,
- summary diagnostics,
- synthetic-gap validation metrics,
- interval coverage,
- actual versus interpolated validation points,
- seasonal validation diagnostics.

Jump-corrected post-gap points are marked in the interpolation plot when jump correction was enabled in network.interpolation.

Usage

```
## S3 method for class 'network_interpolation'
plot(
  x,
  type = c("interpolation", "uncertainty", "availability", "summary", "validation",
    "coverage", "compare", "seasonal_error"),
  series = NULL,
  start = NULL,
  end = NULL,
  show_pi = TRUE,
  show_fit = FALSE,
  free_y = TRUE,
  ncol = 1,
  summary_metric = c("n_imputed", "n_remaining_na", "n_missing_input", "mean_pi_width",
    "median_pi_width", "mean_ref_n", "median_ref_n", "n_gap_jumps_removed",
    "max_abs_gap_jump"),
  validation_metric = c("MAE", "MAPE", "MdAPE", "RMSE", "Bias", "Success_rate",
   "Mean_PI_width", "Mean_ref_n", "End_value_abs_diff", "Max_abs_diff_within_gap"),
  seasonal_metric = c("mean_error", "mean_abs_error", "mean_abs_pct_error", "rmse",
    "coverage"),
  compare_colour = c("series", "gap_steps"),
  facet = TRUE,
  empty = c("plot", "error"),
  ...
)
```

Arguments

x	Object returned by network.interpolation.

type	Character. Plot type. One of "interpolation", "uncertainty", "availability", "summary", "validation", "coverage", "compare", or "seasonal_error".
series	Optional character vector of focal series names to plot.
start	Optional start datetime for time-based plots.
end	Optional end datetime for time-based plots.
show_pi	Logical. If TRUE, show 95% prediction intervals in the interpolation plot where available.
show_fit	Logical. If TRUE, show model-implied fitted values in the interpolation plot where available.
free_y	Logical. If TRUE, facets use free y scales where applicable.
ncol	Integer. Number of facet columns.
summary_metric	Character metric used when type = "summary". One of "n_imputed", "n_remaining_na", "n_missing_input", "mean_pi_width", "median_pi_width", "mean_ref_n", "median_ref_n", "n_gap_jumps_removed", or "max_abs_gap_jump".
validation_metric	Character metric used when type = "validation". One of "MAE", "MAPE", "MdAPE", "RMSE", "Bias", "Success_rate", "Mean_PI_width", "Mean_ref_n", "End_value_abs_diff", or "Max_abs_diff_within_gap".
seasonal_metric	Character metric used when type = "seasonal_error". One of "mean_error", "mean_abs_error", "mean_abs_pct_error", "rmse", or "coverage".
compare_colour	Character. Colouring used when type = "compare". One of "series" or "gap_steps".
facet	Logical. If TRUE, use faceting where supported.
empty	Character. Behavior when no suitable data are available: "plot" returns an informative empty plot and "error" throws an error.
...	Further arguments passed through the generic.

Details

Plot types:

"interpolation" Shows original and interpolated focal series through time. Imputed points are highlighted. If jump correction was applied, corrected post-gap points are marked in green.

"uncertainty" Shows the width of the prediction interval through time.

"availability" Shows the number of valid reference sensors available at each step.

"summary" Shows per-series summary diagnostics produced by the interpolation run.

"validation" Shows synthetic-gap recovery metrics across tested gap lengths.

"coverage" Shows how often true values fall inside the nominal 95% prediction interval across gap lengths.

"compare" Scatter plot of actual versus interpolated values from synthetic-gap validation.

"seasonal_error" Shows how validation error varies through the season, summarized by day of year.

Validation-based plots require that network.interpolation(..., assess = TRUE) was used to create x.

Value

A ggplot2 object.

See Also

network.interpolation

Examples

```
#library(dendRoAnalyst)
#data("gf_nepa17")

#df1 <- gf_nepa17
#df1[40:50, "T2"] <- NA

#ref <- cbind(gf_nepa17, gf_nepa17[, 2:3], gf_nepa17[, 2:3])
#colnames(ref) <- c("Time", "T1", "T2", "T3", "T4", "T5", "T6")

#out <- network.interpolation(
#  df1, ref,
#  niMethod = "proportional",
#  n_boot = 100,
#  assess = TRUE,
#  assess_lengths_steps = c(1, 2, 4),
#  correct_gap_jumps = TRUE,
#  jump_threshold = 0.05
#)

#plot(out)
#plot(out, type = "uncertainty")
#plot(out, type = "availability")
#plot(out, type = "summary", summary_metric = "n_gap_jumps_removed")
#plot(out, type = "validation", validation_metric = "MAPE")
#plot(out, type = "coverage")
#plot(out, type = "compare")
#plot(out, type = "seasonal_error", seasonal_metric = "mean_abs_error")
```

---

plot.SC_output *Plot method for stem-cycle output*

Description

Unified S3 plotting method for objects returned by phase.sc(). Supports raw phase timelines, phase ribbons, transition diagnostics, daily/monthly phase balance, cumulative increment, event frequency, phase heatmaps, and daily/monthly boxplots of phase statistics.

For type = "balance", two plotting modes are available. With balance_mode = "time_of_day", phases are drawn as continuous within-day intervals, and the y-axis represents hour of day from

0 to 24. This makes the timing of shrinkage, expansion, and increment visible. For example, if shrinkage occurs from 00:00 to 05:00, expansion from 05:00 to 15:00, and shrinkage again from 15:00 to 24:00, these intervals are shown as continuous coloured blocks along the y-axis.

With `balance_mode = "duration"`, the older stacked-bar behaviour is used. In that case, the y-axis represents the amount of time spent in each phase. With `temporal = "daily"`, the y-axis is hours per day. With `temporal = "monthly"`, the y-axis is hours per month.

Usage

```
## S3 method for class 'SC_output'
plot(
  x,
  y = NULL,
  DOY = NULL,
  Year = NULL,
  type = c("points", "ribbon", "transition", "balance", "increment", "frequency",
    "heatmap", "boxplot"),
  temporal = c("raw", "daily", "monthly"),
  x_axis = c("time", "doy"),
  balance_mode = c("time_of_day", "duration"),
  stat = c("Duration_h", "Duration_m", "Magnitude", "rate"),
  phase = c("all", "Shrinkage", "Expansion", "Increment"),
  cols = c("#fee8c8", "#fdbb84", "#e34a33"),
  phNames = c("Shrinkage", "Expansion", "Increment"),
  transition_linetype = "dashed",
  transition_alpha = 0.55,
  singleton_as_points = TRUE,
  ...
)
```

Arguments

`x`	Object of class `"SC_output"` returned by `phase.sc()`.
`y`	Unused.
`DOY`	Optional numeric vector of length 2 giving start and end day-of-year.
`Year`	Optional numeric year used together with `DOY`.
`type`	Plot type. One of `"points"`, `"ribbon"`, `"transition"`, `"balance"`, `"increment"`, `"frequency"`, `"heatmap"`, or `"boxplot"`.
`temporal`	Temporal scale: `"raw"`, `"daily"`, or `"monthly"`. For `type = "boxplot"`, use `"daily"` or `"monthly"`. For `type = "balance"`, `"frequency"`, and `"heatmap"`, `"raw"` is internally treated as daily summary and a warning is emitted.
`x_axis`	X-axis style for time-based plots. One of `"time"` or `"doy"`. When `"doy"` is used, timestamps are converted to day-of-year on the x-axis. For `temporal = "monthly"`, `"doy"` is not meaningful and time is used instead.
`balance_mode`	For `type = "balance"`, controls how phase balance is drawn. One of `"time_of_day"` or `"duration"`. `"time_of_day"` draws continuous phase intervals along the y-axis from 0 to 24 h, preserving when phases occurred within each day. `"duration"` draws stacked bars showing total time spent in each phase per day or month.

`stat` For `type = "boxplot"`, one of `"Duration_h"`, `"Duration_m"`, `"Magnitude"`, or `"rate"`.

`phase` For `type = "boxplot"`, one of `"all"`, `"Shrinkage"`, `"Expansion"`, or `"Increment"`.

`cols` Vector of three colours for Shrinkage, Expansion, and Increment.

`phNames` Vector of three labels for the three phase names.

`transition_linetype` Line type used for transition markers in `type = "transition"`.

`transition_alpha` Transparency of transition markers.

`singleton_as_points` Logical. If `TRUE`, the boxplot panel falls back to a point plot when every group contains only one value.

`...` Unused.

Details

Plot types:

- `"points"`: dendrometer series with points coloured by phase.
- `"ribbon"`: dendrometer series with background ribbons for contiguous phases.
- `"transition"`: dendrometer series with vertical lines at phase changes.
- `"balance"`: phase balance through time. With `balance_mode = "time_of_day"`, phase intervals are drawn continuously along the y-axis from 0 to 24 h, so the timing of shrinkage, expansion, and increment within each day is visible. With `balance_mode = "duration"`, stacked bars show total time spent in each phase per day or month.
- `"increment"`: cumulative increment, phase 3, over time.
- `"frequency"`: number of phase events starting per day or month.
- `"heatmap"`: dominant phase by hour-of-day across dates or months.
- `"boxplot"`: distribution of selected cycle statistics after assembling cycle summaries to the chosen daily or monthly time scale.

The `"frequency"` plot uses `SC_cycle`. The `"heatmap"` plot uses `SC_phase`. The `"transition"` plot is especially useful for checking whether smoothing reduced spurious phase switching.

Value

A `ggplot2` object, returned invisibly.

Examples

```
# data(gf_nepa17)
# sc <- phase.sc(df = gf_nepa17, TreeNum = 1, smoothing = 12)

# plot(sc)
# plot(sc, type = "ribbon")
# plot(sc, type = "transition")
```

```
# Daily within-day phase timing
# plot(sc, type = "balance", temporal = "daily")

# Daily within-day phase timing for selected DOY range
# plot(sc, type = "balance", temporal = "daily",
#      DOY = c(150, 160), Year = 2023)

# Old stacked-duration balance plot
# plot(sc, type = "balance", temporal = "daily",
#      balance_mode = "duration")

# Monthly stacked duration
# plot(sc, type = "balance", temporal = "monthly",
#      balance_mode = "duration")

# Boxplots
# plot(sc, type = "boxplot", stat = "Magnitude", temporal = "monthly")
# plot(sc, type = "boxplot", stat = "Duration_h",
#      temporal = "daily", phase = "Shrinkage")
```

`plot.SC_output_clim` *Plot climate-augmented stem-cycle output*

Description

S3 plotting method for objects of class `SC_output_clim`. This allows climate-augmented stem-cycle outputs to be visualized directly with the generic `plot()` function.

Usage

```
## S3 method for class 'SC_output_clim'
plot(
  x,
  y = NULL,
  ...,
  climate_var = NULL,
  climate_vars = NULL,
  numeric_vars = NULL,
  compare = FALSE,
  temporal = NULL
)
```

Arguments

x	An object of class `SC_output_clim`.
y	Optional climate variable name passed as the second argument.

...	Additional arguments passed to dm_plot_climate() or dm_plot_climate_compare().
climate_var	Character. Name of one climate variable to plot.
climate_vars	Character vector of climate variables for comparison plots.
numeric_vars	Character vector of numeric variables used for correlation or regression heatmaps.
compare	Logical. If TRUE, dm_plot_climate_compare() is called.
temporal	Deprecated argument kept for compatibility with older examples. It is ignored.

Value

A ggplot2 object, returned invisibly.

Examples

```
# plot(sc_clim, climate_var = "temp_mean_phase", scale = "cycle",
#      type = "violin")
```

---

plot.summary_mov_cor_dm

Plot method for summaries of moving dendrometer-climate correlation

---

Description

S3 plotting method for objects returned by summary.mov_cor_dm().

Usage

```
## S3 method for class 'summary_mov_cor_dm'
plot(
  x,
  y = NULL,
  type = c("peak_corr", "prop_significant", "peak_time"),
  x_axis = c("time", "doy"),
  low_col = "red",
  mid_col = "white",
  high_col = "blue",
  show_window_label = TRUE,
  ...
)
```

Arguments

x	Object of class "summary_mov_cor_dm".
y	Unused.
type	Plot type. One of "peak_corr", "prop_significant", or "peak_time".
x_axis	X-axis style for type = "peak_time". One of "time" or "doy".
low_col	Colour for negative values.
mid_col	Colour for zero.
high_col	Colour for positive values.
show_window_label	Logical. If TRUE, climate settings are shown as labels in type = "peak_corr".
...	Unused.

Value

A ggplot2 object, returned invisibly.

---

plot.ZG_output *Plot method for zero-growth output*

Description

Unified S3 plotting method for objects returned by phase.zg(). It supports time-series views of GRO and TWD, phase ribbons and transitions, TWD-only plots, ABr summaries, phase-balance views, and boxplots of phase-level statistics. For boxplots, phase summaries are first assembled to the requested daily or monthly time scale, and rows with missing x-axis grouping labels are removed before plotting.

The plotting method operates on the two tables returned by phase.zg():

- ZG_phase: point-level time series with TIME, dm, Phases, TWD, and GRO;
- ZG_cycle: phase-level summaries including max.twd, AUC.load, AUC.total, and ABr.value.

Usage

```
## S3 method for class 'ZG_output'
plot(
  x,
  y = NULL,
  DOY = NULL,
  Year = NULL,
  view = NULL,
 type = c("gro_twd", "phase_ribbon", "transition", "twd", "abr", "phase_summary",
    "balance", "boxplot"),
  temporal = c("raw", "daily", "monthly"),
  x_axis = c("time", "doy"),
```

```
  stat = c("Duration_h", "Magnitude", "rate", "max.twd", "Avg.twd", "STD.twd",
    "AUC.load", "AUC.total", "ABr.value"),
  phase = c("auto", "TWD", "GRO", "all"),
  box_group = c("period", "month_of_year", "doy"),
  twd_fun = c("mean", "max"),
  gro_fun = c("max", "mean"),
  transition_linetype = "dashed",
  transition_alpha = 0.55,
  cols = c(TWD = "red", GRO = "blue"),
  singleton_as_points = TRUE,
  balance_mode = c("time_of_day", "duration"),
  balance_gap = c("carry_forward", "observed_only"),
  ...
)
```

Arguments

x	Object of class "ZG_output" returned by phase.zg().
y	Unused.
DOY	Optional numeric vector of length 2 giving start and end day-of-year for truncating the plotting window.
Year	Optional numeric year used together with DOY.
view	Optional backward-compatible argument. Use "boxplot" to force boxplot mode.
type	Plot type. One of "gro_twd", "phase_ribbon", "transition", "twd", "abr", "phase_summary", "balance", or "boxplot".
temporal	Temporal scale. One of "raw", "daily", or "monthly".
x_axis	X-axis style for time-based plots. One of "time" or "doy". When "doy" is used, timestamps are converted to day-of-year. For temporal = "monthly", "doy" is not meaningful and time is used instead.
stat	For type = "boxplot", one of "Duration_h", "Magnitude", "rate", "max.twd", "Avg.twd", "STD.twd", "AUC.load", "AUC.total", or "ABr.value".
phase	For type = "boxplot", one of "auto", "TWD", "GRO", or "all".
box_group	For type = "boxplot", grouping mode: "period", "month_of_year", or "doy". "period" uses exact dates/months from the already aggregated daily or monthly phase summaries; the other two pool values across repeated seasonal positions. Rows with missing grouping labels are removed before plotting.
twd_fun	Aggregation function for point-level TWD when temporal != "raw". One of "mean" or "max".
gro_fun	Aggregation function for point-level GRO when temporal != "raw". One of "max" or "mean".
transition_linetype	Line type used for transition markers.
transition_alpha	Transparency of transition markers.

cols	Vector of two colours for TWD and GRO. It may be named (c(TWD = "red", GRO = "blue")) or unnamed length 2.
singleton_as_points	Logical. If TRUE, the boxplot panel falls back to a point plot when every group contains only one value.
balance_mode	For type = "balance", one of "time_of_day" or "duration". "time_of_day" draws continuous TWD/GRO intervals along the y-axis from 0 to 24 h, preserving when phases occurred within each day. "duration" draws stacked bars of total hours spent in each phase per day or month.
balance_gap	For type = "balance", one of "carry_forward" or "observed_only". "carry_forward" assigns gaps to the previous observed phase; "observed_only" counts only observed intervals.
...	Unused.

Details

ABr formula shown by this plot method.

When type = "abr" or when stat = "ABr.value" is used in boxplots, the displayed values come directly from phase.zg():

$$ABr = \max(TWD) \times \left(\frac{AUC_{load}}{\max(TWD)} \right)^{\beta}$$

where

$$AUC_{load} = \int_{t_{start}}^{t_{max}} TWD(t)\, dt$$

and

$$AUC_{total} = \int_{t_{start}}^{t_{end}} TWD(t)\, dt.$$

This function does not recompute ABr or AUC values; it only visualizes the values already stored in the ZG_output object.

Aggregation behavior.

For temporal = "daily" or "monthly", point-level series are aggregated before plotting. TWD and GRO can be summarized using "mean" or "max", depending on twd_fun and gro_fun.

For phase-level summaries in ZG_cycle, the plot method aggregates statistics by daily or monthly period. In those aggregated summaries:

- Duration_h, Magnitude, AUC.load, AUC.total, and ABr.value are summed;
- max.twd is taken as the maximum;
- Avg.twd and STD.twd are averaged;
- rate is recalculated from aggregated magnitude and duration.

Phase balance.

For `type = "balance"`, two balance styles are available through `balance_mode`. With `balance_mode = "time_of_day"`, the plot draws TWD and GRO intervals on a 0–24 hour y-axis, preserving the order in which phases occurred within each day, similar to the balance option in `plot.SC_output()`. With `balance_mode = "duration"`, the plot draws stacked bars showing the total number of hours spent in each phase. Missing or irregular intervals can be handled with `balance_gap`.

Boxplot statistics.

The following phase-level variables can be visualized in `type = "boxplot"`. For `box_group = "period"`, boxplots are drawn on a continuous date axis spanning the selected multi-annual data window, so empty days or months remain visible as gaps rather than being removed from the x-axis:

- `"Duration_h"`
- `"Magnitude"`
- `"rate"`
- `"max.twd"`
- `"Avg.twd"`
- `"STD.twd"`
- `"AUC.load"`
- `"AUC.total"`
- `"ABr.value"`

Phase handling.

For `phase = "auto"`, TWD-specific statistics (`max.twd`, `Avg.twd`, `STD.twd`, `AUC.load`, `AUC.total`, `ABr.value`) automatically select TWD phases, whereas GRO-specific variables (`Magnitude`, `rate`) select GRO phases.

Value

A `ggplot2` object, returned invisibly.

Examples

```
# data(gf_nepa17)
# zg <- phase.zg(df = gf_nepa17[1:500, ], TreeNum = 1, beta = 0.1)

# Raw GRO and TWD time series
# plot(zg)

# Daily aggregated series
# plot(zg, temporal = "daily")

# Monthly TWD plot
# plot(zg, temporal = "monthly", type = "twd")

# ABr bar plot
# plot(zg, temporal = "monthly", type = "abr")
```

```
# Daily phase timing, similar to plot.SC_output balance
# plot(zg, type = "balance", temporal = "daily",
#      balance_mode = "time_of_day")

# Stacked phase-duration balance
# plot(zg, type = "balance", temporal = "daily",
#      balance_mode = "duration")

# Transition plot in day-of-year coordinates
# plot(zg, type = "transition", x_axis = "doy",
#      DOY = c(50, 100), Year = 2017)

# Boxplots of peak TWD
# plot(zg, type = "boxplot", stat = "max.twd",
#      temporal = "daily", box_group = "doy")

# Boxplots of AUC.load
# plot(zg, type = "boxplot", stat = "AUC.load",
#      temporal = "monthly", box_group = "month_of_year")

# Boxplots of ABr.value
# plot(zg, type = "boxplot", stat = "ABr.value",
#      temporal = "monthly", box_group = "month_of_year")
# plot(zg, view = "boxplot", stat = "ABr.value", temporal = "monthly")
```

`plot.ZG_output_clim` *Plot climate-augmented zero-growth output*

Description

S3 plotting method for objects of class `ZG_output_clim`. This allows climate-augmented zero-growth outputs to be visualized directly with the generic `plot()` function.

Usage

```
## S3 method for class 'ZG_output_clim'
plot(
  x,
  y = NULL,
  ...,
  climate_var = NULL,
  climate_vars = NULL,
  numeric_vars = NULL,
  compare = FALSE,
  temporal = NULL
)
```

Arguments

x	An object of class ZG_output_clim.
y	Optional climate variable name passed as the second argument.
...	Additional arguments passed to dm_plot_climate() or dm_plot_climate_compare().
climate_var	Character. Name of one climate variable to plot.
climate_vars	Character vector of climate variables for comparison plots.
numeric_vars	Character vector of numeric variables used for correlation or regression heatmaps.
compare	Logical. If TRUE, dm_plot_climate_compare() is called.
temporal	Deprecated argument kept for compatibility with older examples. It is ignored.

Value

A ggplot2 object, returned invisibly.

Examples

```
# plot(zg_clim, climate_var = "VPD_mean_phase", scale = "cycle",
#      type = "boxplot")
```

---

plot_dm_assessment *Plot interpolation assessment metrics*

Description

Plot interpolation assessment metrics

Usage

```
plot_dm_assessment(x, metric = "MdAPE", series = NULL, facet = TRUE)
```

Arguments

x	Object returned by dm.na.interpolation() with assess = TRUE.
metric	Character. One of: "MAPE", "MdAPE", "RMSE_pct", "Bias_pct", "Max_diff_abs", "Min_diff_abs", "Time_max_diff_h", "Time_min_diff_h".
series	Optional character vector of series names to plot.
facet	Logical. If TRUE, create one panel per series.

Value

A ggplot2 object.

Examples

```
#res <- dm.na.interpolation(
#  nepa17[1:1000, ],
#  resolution = 60,
#  fill = TRUE,
#  method = "seasonal",
#  assess = TRUE
#)
#plot_dm_assessment(res, metric = "MdAPE")
```

---

plot_dm_gaps *Plot detected gaps in dendrometer data*

Description

Plot detected gaps in dendrometer data

Usage

```
plot_dm_gaps(x, series = NULL, empty_gaps = c("plot", "error"))
```

Arguments

x	Object returned by dm.na.interpolation().
series	Optional character vector of series names to plot.
empty_gaps	Character. Behavior when no gaps are present: "plot" for an informative empty plot, or "error".

Value

A ggplot2 object.

Examples

```
#res <- dm.na.interpolation(nepa17[1:1000, ], resolution = 60)
#plot_dm_gaps(res)
```

plot_dm_interpolation *Plot original and interpolated dendrometer series*

Description

Plot original and interpolated dendrometer series

Usage

```
plot_dm_interpolation(
  x,
  original = NULL,
  series = NULL,
  start = NULL,
  end = NULL,
  free_y = TRUE,
  ncol = 1
)
```

Arguments

x	Object returned by dm.na.interpolation().
original	Optional original input data frame used in dm.na.interpolation().
series	Optional character vector of series names to plot.
start	Optional start datetime for plotting subset.
end	Optional end datetime for plotting subset.
free_y	Logical. If TRUE, each facet gets its own y-scale.
ncol	Integer. Number of facet columns.

Value

A ggplot2 object.

Examples

```
#res <- dm.na.interpolation(
#  nepa17[1:1000, ],
#  resolution = 60,
#  fill = TRUE,
#  method = "spline"
#)
#plot_dm_interpolation(res, original = nepa17[1:1000, ])
```

plot_event_climate_box *Plot event-based climate distributions*

Description

Draws boxplots or violins of event-based climate variables across phases or event types, with optional significance annotation.

Usage

```
plot_event_climate_box(
  event_data,
  climate_var,
  group_var = c("Phase", "event_type"),
  facet_by = c("none", "month", "month_of_year", "year"),
  Year = NULL,
  DOY = NULL,
  geom = c("boxplot", "violin", "both"),
  add_test = TRUE,
  test_method = c("auto", "anova", "kruskal", "t.test", "wilcox"),
  point_alpha = 0.5
)
```

Arguments

event_data	Output of dm_event_climate().
climate_var	Name of the climate-derived column to plot.
group_var	Grouping variable. One of "Phase" or "event_type".
facet_by	One of "none", "month", "month_of_year", or "year".
Year	Optional numeric year or vector of years for filtering.
DOY	Optional numeric vector of length 2 for filtering.
geom	One of "boxplot", "violin", or "both".
add_test	Logical. If TRUE, adds per-panel test results.
test_method	Passed to dm_event_climate_test().
point_alpha	Point transparency.

Value

A ggplot2 object.

Examples

```
data(gf_nepa17)
data(ktm_clim_hourly)

zg <- phase.zg(df = gf_nepa17[1:800, ], TreeNum = 1)
evt_zg <- dm_event_climate(
  zg,
  ktm_clim_hourly,
  event = "phase_start",
  windows = c(0, 3, 6, 12, 24),
  mean_vars = c("temp", "VPD", "RH"),
  max_vars  = c("VPD"),
  sum_vars  = c("prec")
)

if (any(!is.na(evt_zg$VPD_mean_prev_6h))) {
  plot_event_climate_box(
    evt_zg,
    climate_var = "VPD_mean_prev_6h",
    group_var = "Phase",
    facet_by = "month_of_year",
    geom = "both",
    add_test = TRUE
  )
}
```

`plot_event_climate_relation` *Plot event-based climate-response relationships*

Description

Draws a scatterplot between an event-based climate variable and a response variable and annotates the fitted linear relationship with R^2, slope, and p-value.

Usage

```
plot_event_climate_relation(
  event_data,
  climate_var,
  response_var,
  group_var = NULL,
  Year = NULL,
  DOY = NULL,
  add_smooth = TRUE,
  point_alpha = 0.7
)
```

Arguments

event_data	Output of dm_event_climate().
climate_var	Name of the climate-derived column.
response_var	Name of the response variable.
group_var	Optional grouping variable for color.
Year	Optional numeric year or vector of years for filtering.
DOY	Optional numeric vector of length 2 for filtering.
add_smooth	Logical. If TRUE, adds linear fit line.
point_alpha	Point transparency.

Value

A ggplot2 object.

Examples

```
data(gf_nepa17)
data(ktm_clim_hourly)

zg <- phase.zg(df = gf_nepa17[1:800, ], TreeNum = 1)
evt_maxtwd <- dm_event_climate(
  zg,
  ktm_clim_hourly,
  event = "max_twd",
  windows = c(0, 3, 6, 12, 24),
  mean_vars = c("temp", "VPD", "RH"),
  max_vars  = c("VPD"),
  sum_vars  = c("prec")
)

if (all(c("VPD_mean_prev_6h", "max.twd") %in% names(evt_maxtwd)) &&
    any(is.finite(evt_maxtwd$VPD_mean_prev_6h)) &&
    any(is.finite(evt_maxtwd$max.twd))) {
  plot_event_climate_relation(
    evt_maxtwd,
    climate_var = "VPD_mean_prev_6h",
    response_var = "max.twd",
    group_var = "Phase"
  )
}
```

plot_mov.cor *Backward-compatible wrapper for plotting moving correlation*

Description

Wrapper around `plot()` for objects returned by `mov.cor.dm()`.

Usage

```
plot_mov.cor(mov.cor.output, ...)
```

Arguments

`mov.cor.output`	Output of `mov.cor.dm()`.
`...`	Passed to `plot.mov_cor_dm()`.

Value

A `ggplot2` object.

print.dm_growth_fit *Print a dm_growth_fit object*

Description

Prints a compact overview of an object returned by [dm.growth.fit()] or [dm.growth.fit.double()].

Usage

```
## S3 method for class 'dm_growth_fit'
print(x, ...)
```

Arguments

`x`	An object of class `"dm_growth_fit"`.
`...`	Further arguments passed to or from other methods.

Value

The input object, invisibly.

print.mov_cor_dm *Print method for running dendrometer-climate correlation objects*

Description

Print method for running dendrometer-climate correlation objects

Usage

```
## S3 method for class 'mov_cor_dm'
print(x, ...)
```

Arguments

x	Object of class "mov_cor_dm".
...	Unused.

Value

The object, invisibly.

print.summary.clim_twd_stats *Print summary of clim.twd.stats output*

Description

Print summary of clim.twd.stats output

Usage

```
## S3 method for class 'summary.clim_twd_stats'
print(x, ...)
```

Arguments

x	An object of class "summary.clim_twd_stats".
...	Further arguments passed to or from other methods.

Value

The input object, invisibly.

print.summary.clim_twd_test

Print summary of clim.twd.test output

Description

Print summary of clim.twd.test output

Usage

```
## S3 method for class 'summary.clim_twd_test'
print(x, ...)
```

Arguments

x	An object of class "summary.clim_twd_test".
...	Further arguments passed to or from other methods.

Value

The input object, invisibly.

print.summary.dm_epoch

Print summary of a dm_epoch object

Description

Print summary of a dm_epoch object

Usage

```
## S3 method for class 'summary.dm_epoch'
print(x, ...)
```

Arguments

x	Object of class "summary.dm_epoch".
...	Unused.

Value

The input object, invisibly.

`print.summary.dm_growth_fit` *Print a summary.dm_growth_fit object*

Description

Prints a formatted summary of a `"summary.dm_growth_fit"` object.

Usage

```
## S3 method for class 'summary.dm_growth_fit'
print(x, ...)
```

Arguments

`x`	An object of class `"summary.dm_growth_fit"`.
`...`	Further arguments passed to or from other methods.

Value

The input object, invisibly.

`print.summary.dm_wavelet_reconstruct` *Print method for summary.dm_wavelet_reconstruct*

Description

Print method for summary.dm_wavelet_reconstruct

Usage

```
## S3 method for class 'summary.dm_wavelet_reconstruct'
print(x, digits = 4, ...)
```

Arguments

`x`	An object of class `"summary.dm_wavelet_reconstruct"`.
`digits`	Number of digits for rounded numeric printing.
`...`	Further arguments passed to or from other methods.

Value

The input object, invisibly.

print.summary_mov_cor_dm *Print method for summaries of running dendrometer-climate correlation objects*

Description

Print method for summaries of running dendrometer-climate correlation objects

Usage

```
## S3 method for class 'summary_mov_cor_dm'
print(x, ...)
```

Arguments

x	Object of class "summary_mov_cor_dm".
...	Unused.

Value

The summary object, invisibly.

read.climate *Read and standardize climate data for dendrometer analyses*

Description

A robust climate-data reader designed to be as flexible as read.dendrometer(). It accepts data frames and common file formats, auto-detects separators and decimal marks, parses many datetime formats, supports Excel serial dates, supports separate date + time columns, detects the time column automatically, sorts timestamps, removes duplicates, converts numeric-like climate variables, and returns a standardized tibble with a POSIXct TIME column.

Usage

```
read.climate(
  x,
  time_col = NULL,
  vars = NULL,
  sep = NULL,
  dec = NULL,
  header = TRUE,
  sheet = 1,
  tz = "UTC",
```

```
  drop_duplicate_time = TRUE,
  min_time_success = 0.6,
  verbose = TRUE,
  datetime_col = NULL,
  date_col = NULL,
  range = NULL,
  na = c("", "NA", "NaN", "nan", "null", "NULL", "-9999", "-999", "N/A"),
  assume_midnight = TRUE,
  orders = NULL,
  excel_dates = c("auto", "none", "1900", "1904"),
  drop_empty_cols = TRUE,
  trim_names = TRUE,
  detect_resolution = FALSE,
  return_report = FALSE,
  quiet = !verbose
)
```

Arguments

x	A data frame or path to a file. Supported file extensions are csv, txt, tsv, tab, dat, xls, xlsx, rds, rda, and RData.
time_col	Backward-compatible explicit time/datetime column name or index. If date_col is also supplied, this is treated as the time-of-day column.
vars	Optional character vector of climate variables to keep.
sep	Optional field separator for text files. If NULL, it is auto-detected.
dec	Optional decimal mark for text files. If NULL, it is auto-detected.
header	Logical; passed to text-file readers.
sheet	Sheet name or index for Excel files.
tz	Time zone for parsed timestamps.
drop_duplicate_time	Logical; if TRUE, duplicated timestamps are removed.
min_time_success	Minimum parsing success proportion for automatic time detection.
verbose	Logical; print an import summary.
datetime_col	Optional explicit datetime column name or index. Prefer this when the file has one combined timestamp column.
date_col	Optional explicit date column name or index. Can be combined with time_col, or parsed alone with midnight appended when assume_midnight = TRUE.
range	Optional Excel cell range.
na	Strings to treat as missing values.
assume_midnight	Logical; if TRUE, date-only values are assigned 00:00:00.
orders	Optional lubridate parse_date_time() orders.
excel_dates	One of auto, none, 1900, or 1904.

drop_empty_cols
: Logical; if TRUE, columns that are completely empty are removed.

trim_names
: Logical; if TRUE, trim whitespace from column names.

detect_resolution
: Logical; if TRUE, attach simple time-resolution diagnostics.

return_report
: Logical; if TRUE, return list(data = ..., report = ...).

quiet
: Logical; suppress messages. By default this is the inverse of verbose.

Value

A tibble of class dm_clim with TIME in the first column. An import report is attached as attr(x, "import_report").

---

read.dendrometer *Reading dendrometer data*

Description

Reads dendrometer data from `.csv`, `.txt`, `.tsv`, or `.xlsx` files, automatically parsing the datetime information and converting it to `%Y-%m-%d %H:%M:%S` in the requested timezone.

Supports:

- automatic delimiter detection for text files,
- real decimal-mark auto-detection for text files,
- Excel serial date support (`1900` / `1904` systems),
- separate `date_col` + `time_col`,
- optional import report output,
- optional time-resolution diagnostics.

Usage

```
read.dendrometer(
  file,
  sep = NULL,
  dec = NULL,
  datetime_col = 1,
  date_col = NULL,
  time_col = NULL,
  tz = "UTC",
  sheet = NULL,
  range = NULL,
  na = c("", "NA", "NaN", "nan", "null", "NULL", "-9999"),
  assume_midnight = TRUE,
  orders = NULL,
  excel_dates = c("auto", "none", "1900", "1904"),
```

```
  drop_dup_times = TRUE,
  detect_resolution = FALSE,
  return_report = FALSE,
  quiet = TRUE
)
```

Arguments

file	Path to file (.csv, .txt, .tsv, .xlsx).
sep	Optional delimiter for text files. If NULL, auto-detect among comma, semicolon, tab, and pipe.
dec	Optional decimal mark for text files. If NULL, auto-detect between "." and ",".
datetime_col	Integer or name of the datetime column (default 1). Ignored if date_col is provided.
date_col	Optional integer or name of a date column.
time_col	Optional integer or name of a time column. Used together with date_col. If NULL, only the date column is parsed and "00:00:00" can be appended if assume_midnight = TRUE.
tz	Time zone for parsed datetimes (default "UTC").
sheet	Excel sheet name or index (for .xlsx; default NULL = first sheet).
range	Excel cell range (optional).
na	Character vector of strings to treat as NA.
assume_midnight	Logical; if TRUE, rows with only a date get "00:00:00".
orders	Optional vector of **lubridate** orders to try. If NULL, a comprehensive default set is used.
excel_dates	Character. One of "auto", "none", "1900", or "1904". Controls handling of numeric Excel serial dates.
drop_dup_times	Logical; if TRUE, drop duplicated timestamps (keep first) with a warning/message.
detect_resolution	Logical; if TRUE, compute basic time-resolution diagnostics.
return_report	Logical; if TRUE, return a list with $data and $report. If FALSE, return the tibble only. In both cases, the report is attached as attr(x, "import_report").
quiet	Logical; if TRUE, suppress informational messages.

Value

If return_report = FALSE, a tibble with a POSIXct first column and the remaining data columns unchanged.

If return_report = TRUE, a list with:

$data The imported tibble.

$report A structured import report.

reso_dm *Detect Temporal Resolution and Irregularities in Time Series*

Description

Determines the average temporal resolution (in minutes) of a time series vector (e.g., dendrometer timestamps) and detects inconsistencies in time intervals. If the time column is character-formatted, it is automatically converted to POSIXct. Any inconsistent intervals are flagged and printed.

Usage

```
reso_dm(input_time)
```

Arguments

input_time	A vector of class POSIXct, Date, or character representing time stamps.

Details

This function is helpful for checking if a time series (especially dendrometer data) is regularly sampled. It handles both regular and irregular timestamps and gives feedback if the resolution changes.

Value

A single integer: the estimated (rounded) average resolution in minutes. If multiple intervals are detected, a warning and index positions are printed.

Examples

```
## Not run:
# Regular 30-minute time sequence
time_seq <- seq.POSIXt(from = as.POSIXct("2023-06-01 00:00:00"),
                       by = "30 min", length.out = 100)
reso_dm(time_seq)  # Should return 30

# With character time input
time_char <- format(time_seq, format = "%Y-%m-%d %H:%M:%S")
reso_dm(time_char)  # Auto converts to POSIXct

# Introduce an irregular step
time_seq[51] <- time_seq[50] + 60  # One-time 1-hour jump
reso_dm(time_seq)  # Should print warning and irregular step index

## End(Not run)
```

smooth_dm *Smoothing of Dendrometer Time Series*

Description

Applies various smoothing techniques to dendrometer (dm) time series data using a user-defined or automatically detected temporal resolution. The function supports several smoothing methods: robust median+mean, penalized spline, Savitzky-Golay filter, exponential moving average (EMA), and LOESS.

Usage

```
smooth_dm(
  time,
  dm,
  resolution_min = NULL,
  method = c("median_mean", "pspline", "sg", "ema", "loess"),
  window_hours = 3,
  sg_order = 2,
  ema_alpha = NULL
)
```

Arguments

time	A POSIXct vector representing the time column.
dm	A numeric vector of dendrometer values corresponding to time.
resolution_min	Integer. The resolution of the time series in minutes. If NULL, the resolution is auto-detected based on median time difference.
method	Smoothing method. One of: "median_mean", "pspline", "sg" (Savitzky-Golay), "ema" (exponential moving average), or "loess".
window_hours	Numeric. Smoothing window length in hours. Converted to points using resolution.
sg_order	Integer. Polynomial order for Savitzky-Golay smoothing. Ignored unless method = "sg".
ema_alpha	Numeric. Smoothing factor (0–1) for exponential moving average. If NULL, it is automatically calculated from the window size.

Details

The function is designed to smooth dendrometer time series (e.g., 10–60 min resolution) while preserving key features such as diurnal fluctuations or long-term growth, depending on the window size. It auto-detects the resolution (in minutes) if not provided.

Value

A numeric vector of the same length as dm containing the smoothed values.

Examples

```
## Not run:
# Example: Create synthetic dendrometer time series (30-min resolution)
time_seq <- seq.POSIXt(from = as.POSIXct("2023-06-01 00:00:00"),
                       to   = as.POSIXct("2023-06-03 00:00:00"),
                       by   = "30 mins")
set.seed(123)
dm_raw <- cumsum(rnorm(length(time_seq), mean = 0.005, sd = 0.01)) +
       0.1 * sin(2 * pi * as.numeric(difftime(time_seq, min(time_seq), units = "hours")) / 24)

# Median plus moving mean smoothing (default robust filter)
dm_medmean <- smooth_dm(time = time_seq, dm = dm_raw,
                        method = "median_mean", window_hours = 3)

# Penalized spline smoothing
dm_pspline <- smooth_dm(time = time_seq, dm = dm_raw,
                        method = "pspline", window_hours = 6)

# Savitzky-Golay filter (requires 'signal' package)
if (requireNamespace("signal", quietly = TRUE)) {
  dm_sg <- smooth_dm(time = time_seq, dm = dm_raw,
                     method = "sg", window_hours = 2, sg_order = 2)
}

# Exponential moving average smoothing
dm_ema <- smooth_dm(time = time_seq, dm = dm_raw,
                    method = "ema", window_hours = 4)

# LOESS smoothing
dm_loess <- smooth_dm(time = time_seq, dm = dm_raw,
                      method = "loess", window_hours = 3)

# Plot raw and smoothed series
plot(time_seq, dm_raw, type = "l", col = "gray", lwd = 1, main = "Smoothed Dendrometer Series",
     ylab = "DM", xlab = "Time")
lines(time_seq, dm_medmean, col = "blue", lwd = 2)
lines(time_seq, dm_pspline, col = "green", lwd = 2)
lines(time_seq, dm_ema, col = "orange", lwd = 2)
lines(time_seq, dm_loess, col = "purple", lwd = 2)
legend("topright", legend = c("Raw", "Median+Mean", "P-spline", "EMA", "LOESS"),
       col = c("gray", "blue", "green", "orange", "purple"), lwd = 2)

## End(Not run)
```

`summary.clim_twd_stats`

Summarize clim.twd.stats output

Description

Summarize clim.twd.stats output

Usage

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

Arguments

object	An object of class "clim_twd_stats".
...	Further arguments passed to or from other methods.

Value

An object of class "summary.clim_twd_stats".

---

summary.clim_twd_test *Summarize clim.twd.test output*

Description

Summarize clim.twd.test output

Usage

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

Arguments

object	An object of class "clim_twd_test".
...	Further arguments passed to or from other methods.

Value

An object of class "summary.clim_twd_test".

summary.dm_epoch *Summarize a dm_epoch object*

Description

Summarize a dm_epoch object

Usage

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

Arguments

object	Object of class "dm_epoch".
...	Unused.

Value

An object of class "summary.dm_epoch".

summary.dm_growth_fit *Summarize a dm_growth_fit object*

Description

Summarizes an object returned by [dm.growth.fit()] or [dm.growth.fit.double()].

Usage

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

Arguments

object	An object of class "dm_growth_fit".
...	Further arguments passed to or from other methods.

Value

An object of class "summary.dm_growth_fit".

summary.dm_wavelet *Summarize a dm_wavelet object*

Description

Summarizes wavelet-analysis results produced by `dm_wavelet()`.

For each analyzed series, the summary reports:

- the dominant period in hours,
- the corresponding maximum average wavelet power,
- the analyzed period range in hours,
- the detected temporal resolution.

Usage

```
## S3 method for class 'dm_wavelet'
summary(object, top_n = 3, ...)
```

Arguments

`object`	An object of class `"dm_wavelet"`.
`top_n`	Integer. Number of strongest periods to report per series.
`...`	Further arguments passed to or from other methods.

Value

An object of class `"summary.dm_wavelet"`.

summary.dm_wavelet_reconstruct *Summarize a dm_wavelet_reconstruct object*

Description

Summarizes the output of `dm_wavelet_reconstruct()`.

For each reconstructed series, the summary reports:

- number of observations,
- variance of the original series,
- variance of the reconstructed component,
- variance of the remaining component,
- variance of the returned filtered series,
- proportion of original variance represented by reconstructed and filtered series,
- correlation and R^2 between original and reconstructed,
- number and range of periods used in reconstruction.

Usage

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

Arguments

object	An object of class "dm_wavelet_reconstruct".
...	Further arguments passed to or from other methods.

Value

An object of class "summary.dm_wavelet_reconstruct" with elements:

overview One-row summary of the reconstruction object.

series_summary Per-series summary table.

used_periods The table of periods actually used in the reconstruction.

selection The selection settings used for reconstruction.

---

summary.mov_cor_dm *Summary method for running dendrometer-climate correlation objects*

Description

Summarizes running dendrometer-climate correlations for both bootstrapped and non-bootstrapped outputs, and appends climate-specific settings to the summary tables.

Usage

```
## S3 method for class 'mov_cor_dm'
summary(object, absolute = TRUE, top_n = 5, ...)
```

Arguments

object	Object of class "mov_cor_dm".
absolute	Logical. If TRUE (default), the strongest correlation is identified by absolute magnitude.
top_n	Integer. Number of top windows to return per climate variable.
...	Unused.

Value

A list of class "summary_mov_cor_dm" containing:

- table: summary table for each climate variable
- top_windows: top running windows per climate variable
- metadata: metadata from the original object
- call: original function call

`twd.maxima` *Locating the maxima of TWD periods*

Description

This function detects the TWD phases, including their beginning (TWDb), using the phase.zg function. Then it calculates the number, time of occurance (Tm) and value of every local maximum within each TWD phase. In addition it calculates the time difference between 'TWDb' and each 'Tm' within each TWD phase.

Usage

```
twd.maxima(df, TreeNum, smoothing = 5)
```

Arguments

`df`	data frame with first column containing date and time in the format `yyyy-mm-dd HH:MM:SS`. It should contain data with constant temporal resolution for best results.
`TreeNum`	numerical value indicating the tree to be analysed. E.g. '1' refers to the first dendrometer data column in *df*.
`smoothing`	numerical value from 1 to 12 which indicates the length of the smoothing spline, i.e. 1 = 1 hour and 12 = 12 hours. Default is 5.

Value

A data frame with statistics of maxima in each TWD phase.

Examples

```
library(dendRoAnalyst)
data(gf_nepa17)
df1=gf_nepa17[2500:3500,]
twd_max<-twd.maxima(df=df1, TreeNum=2)
head(twd_max,10)
```

Index