Package 'chillR'

December 11, 2025

Type Package

Title Statistical Methods for Phenology Analysis in Temperate Fruit Trees

Version 0.77

Date 2025-12-10

Description The phenology of plants (i.e. the timing of their annual life phases) depends on climatic cues. For temperate trees and many other plants, spring phases, such as leaf emergence and flowering, have been found to result from the effects of both cool (chilling) conditions and heat. Fruit tree scientists (pomologists) have developed some metrics to quantify chilling and heat (e.g. see Luedeling (2012) <doi:10.1016/j.scienta.2012.07.011>). 'chillR' contains functions for processing temperature records into chilling (Chilling Hours, Utah Chill Units and Chill Portions) and heat units (Growing Degree Hours). Regarding chilling metrics, Chill Portions are often considered the most promising, but they are difficult to calculate. This package makes it easy. 'chillR' also contains procedures for conducting a PLS analysis relating phenological dates (e.g. bloom dates) to either mean temperatures or mean chill and heat accumulation rates, based on long-term weather and phenology records (Luedeling and Gassner (2012) <doi:10.1016/j.agrformet.2011.10.020>). As of version 0.65, it also includes functions for generating weather scenarios with a weather generator, for conducting climate change analyses for temperature-based climatic metrics and for plotting results from such analyses. Since version 0.70, 'chillR' contains a function for interpolating hourly temperature records.

Depends R (>= 3.5.0)

Imports assertthat, dplyr, ecmwfr, fields, GenSA, ggplot2, graphics, grDevices, httr, jsonlite, lubridate, magrittr, metR, patchwork, pls, plyr, progress, purrr, R.utils, raster, Rcpp, RCurl, readxl, reshape2, rlang, RMAWGEN, scales, stats, stringr, tidyr, utils, XML

LinkingTo Rcpp

Suggests knitr, ncdf4, rmarkdown, testthat

VignetteBuilder knitr

License GPL-3

LazyData TRUE

RoxygenNote 7.3.3

Encoding UTF-8

NeedsCompilation yes

Author Eike Luedeling [aut, cre] (ORCID:
<https://orcid.org/0000-0002-7316-3631>),
Lars Caspersen [aut] (ORCID: <https://orcid.org/0009-0000-3057-7327>),
Eduardo Fernandez [aut] (ORCID:
<https://orcid.org/0000-0002-6949-9685>)

Maintainer Eike Luedeling <eike@eikeluedeling.com>

Repository CRAN

Date/Publication 2025-12-11 06:10:44 UTC

Contents

add_date *Add date/time column to data.frame*

Description

Takes the ‘Year‘, ‘Month‘, ‘Day‘ and, if available, ‘Hour‘, ‘Minute‘ and ‘Second‘ columns of a data.frame and uses them to produce a ‘Date‘ column that uses R’s standard data/time format.

Usage

```
add_date(df)
```

Arguments

df	Data frame containing columns ‘Year‘, ‘Month‘, ‘Day‘ and - optionally - ‘Hour‘, ‘Minute‘ and/or ‘Second‘

Details

Converts YEARMODA to R date

Value

data.frame consisting of the df input and a new column ‘Date‘

Author(s)

Eike Luedeling

Examples

```
add_date(KA_weather)
add_date(Winters_hours_gaps)
```

`bloom_prediction` *Bloom prediction from chilling and forcing requirements, assumed to be fulfilled strictly in sequence*

Description

This is a pretty rudimentary function to predict phenological dates from chilling and forcing requirements and hourly chilling and forcing data. Note that there are enormous uncertainties in these predictions, which are hardly ever acknowledged. So please use this function with caution.

Usage

```
bloom_prediction(
  HourChillTable,
  Chill_req,
  Heat_req,
  Chill_model = "Chill_Portions",
  Heat_model = "GDH",
  Start_JDay = 305
)
```

Arguments

`HourChillTable`	a data frame resulting from the chilling_hourtable function.
`Chill_req`	numeric parameter indicating the chilling requirement of the particular growth stage (in the unit specified by "Chill_model")
`Heat_req`	numeric parameter indicating the heat requirement of the particular growth stage (in Growing Degree Hours)
`Chill_model`	character string specifying the chill model to use. This has to correspond to the name of the column in HourChillTable that contains the chill accumulation (e.g "Chilling_Hours", "Chill_Portions" and "Chill_Units").
`Heat_model`	character string specifying the heat model to use. This has to correspond to the name of the column in HourChillTable that contains the heat accumulation (e.g "GDH").
`Start_JDay`	numeric parameter indicating the day when chill accumulation is supposed to start

Details

This function is a bit preliminary at the moment. It will hopefully be refined later.

Chill metrics are calculated as given in the references below. Chilling Hours are all hours with temperatures between 0 and 7.2 degrees C. Units of the Utah Model are calculated as suggested by Richardson et al. (1974) (different weights for different temperature ranges, and negation of chilling by warm temperatures). Chill Portions are calculated according to Fishman et al. (1987a,b). More honestly, they are calculated according to an Excel sheet produced by Amnon Erez and colleagues,

which converts the complex equations in the Fishman papers into relatively simple Excel functions. These were translated into R. References to papers that include the full functions are given below. Growing Degree Hours are calculated according to Anderson et al. (1986), using the default values they suggest.

Value

data frame containing the predicted dates of chilling requirement fulfillment and timing of the phenological stage. Columns are Creqfull, Creq_year, Crey_month, Creq_day and Creq_JDay (the row number, date and Julian date of chilling requirement fulfillement), Hreqfull, Hreq_year, Hreq_month, Hreq_day and Hreq_JDay (the row number, date and Julian date of heat requirement fulfillment - this corresponds to the timing of the phenological event.

Note

After doing extensive model comparisons, and reviewing a lot of relevant literature, I do not recommend using the Chilling Hours or Utah Models, especially in warm climates! The Dynamic Model (Chill Portions), though far from perfect, seems much more reliable.

Author(s)

Eike Luedeling

References

Model references:

Chilling Hours:

Weinberger JH (1950) Chilling requirements of peach varieties. Proc Am Soc Hortic Sci 56, 122-128

Bennett JP (1949) Temperature and bud rest period. Calif Agric 3 (11), 9+12

Utah Model:

Richardson EA, Seeley SD, Walker DR (1974) A model for estimating the completion of rest for Redhaven and Elberta peach trees. HortScience 9(4), 331-332

Dynamic Model:

Erez A, Fishman S, Linsley-Noakes GC, Allan P (1990) The dynamic model for rest completion in peach buds. Acta Hortic 276, 165-174

Fishman S, Erez A, Couvillon GA (1987a) The temperature dependence of dormancy breaking in plants - computer simulation of processes studied under controlled temperatures. J Theor Biol 126(3), 309-321

Fishman S, Erez A, Couvillon GA (1987b) The temperature dependence of dormancy breaking in plants - mathematical analysis of a two-step model involving a cooperative transition. J Theor Biol 124(4), 473-483

Growing Degree Hours:

Anderson JL, Richardson EA, Kesner CD (1986) Validation of chill unit and flower bud phenology models for 'Montmorency' sour cherry. Acta Hortic 184, 71-78

Model comparisons and model equations:

Luedeling E, Zhang M, Luedeling V and Girvetz EH, 2009. Sensitivity of winter chill models for fruit and nut trees to climatic changes expected in California's Central Valley. Agriculture, Ecosystems and Environment 133, 23-31

Luedeling E, Zhang M, McGranahan G and Leslie C, 2009. Validation of winter chill models using historic records of walnut phenology. Agricultural and Forest Meteorology 149, 1854-1864

Luedeling E and Brown PH, 2011. A global analysis of the comparability of winter chill models for fruit and nut trees. International Journal of Biometeorology 55, 411-421

Luedeling E, Kunz A and Blanke M, 2011. Mehr Chilling fuer Obstbaeume in waermeren Wintern? (More winter chill for fruit trees in warmer winters?). Erwerbs-Obstbau 53, 145-155

Review on chilling models in a climate change context:

Luedeling E, 2012. Climate change impacts on winter chill for temperate fruit and nut production: a review. Scientia Horticulturae 144, 218-229

The PLS method is described here:

Luedeling E and Gassner A, 2012. Partial Least Squares Regression for analyzing walnut phenology in California. Agricultural and Forest Meteorology 158, 43-52.

Wold S (1995) PLS for multivariate linear modeling. In: van der Waterbeemd H (ed) Chemometric methods in molecular design: methods and principles in medicinal chemistry, vol 2. Chemie, Weinheim, pp 195-218.

Wold S, Sjostrom M, Eriksson L (2001) PLS-regression: a basic tool of chemometrics. Chemometr Intell Lab 58(2), 109-130.

Mevik B-H, Wehrens R, Liland KH (2011) PLS: Partial Least Squares and Principal Component Regression. R package version 2.3-0. http://CRAN.R-project.org/package0pls.

Some applications of the PLS procedure:

Luedeling E, Kunz A and Blanke M, 2013. Identification of chilling and heat requirements of cherry trees - a statistical approach. International Journal of Biometeorology 57,679-689.

Yu H, Luedeling E and Xu J, 2010. Stronger winter than spring warming delays spring phenology on the Tibetan Plateau. Proceedings of the National Academy of Sciences (PNAS) 107 (51), 22151-22156.

Yu H, Xu J, Okuto E and Luedeling E, 2012. Seasonal Response of Grasslands to Climate Change on the Tibetan Plateau. PLoS ONE 7(11), e49230.

The exact procedure was used here:

Luedeling E, Guo L, Dai J, Leslie C, Blanke M, 2013. Differential responses of trees to temperature variation during the chilling and forcing phases. Agricultural and Forest Meteorology 181, 33-42.

The chillR package:

Luedeling E, Kunz A and Blanke M, 2013. Identification of chilling and heat requirements of cherry trees - a statistical approach. International Journal of Biometeorology 57,679-689.

Examples

```
hourtemps <- stack_hourly_temps(fix_weather(KA_weather[which(KA_weather$Year > 2008), ]),
                                latitude=50.4)

CT <- chilling_hourtable(hourtemps, Start_JDay = 305)
```

```
bloom_prediction(CT, Chill_req = 60, Heat_req = 5000, Chill_model = "Chill_Portions",
                 Heat_model = "GDH", Start_JDay = 305)
```

`bloom_prediction2` *Bloom prediction from chilling and forcing requirements, assumed to be fulfilled strictly in sequence - version 2*

Description

This is a pretty rudimentary function to predict phenological dates from chilling and forcing requirements and hourly chilling and forcing data. Note that there are enormous uncertainties in these predictions, which are hardly ever acknowledged. So please use this function with caution.

Usage

```
bloom_prediction2(
  HourChillTable,
  Chill_req,
  Heat_req,
  permutations = FALSE,
  Chill_model = "Chill_Portions",
  Heat_model = "GDH",
  Start_JDay = 305,
  infocol = NULL
)
```

Arguments

`HourChillTable`	a data frame resulting from the chilling_hourtable function.
`Chill_req`	numeric vector indicating one or multiple chilling requirements of the particular growth stage (in the unit specified by "Chill_model")
`Heat_req`	numeric vector indicating one or multiple heat requirements of the particular growth stage (in Growing Degree Hours)
`permutations`	boolean parameter indicating whether all possible combinations of the supplied chilling and heat requirements should be used. Defaults to FALSE, which means that the function matches chilling and heat requirements according to their positions in the Chill_req and Heat_req vectors and only predicts stage occurrence dates for these combinations.
`Chill_model`	character string specifying the chill model to use. This has to correspond to the name of the column in HourChillTable that contains the chill accumulation (default is "Chill_Portions" for units of the Dynamic Model).
`Heat_model`	character string specifying the heat model to use. This has to correspond to the name of the column in HourChillTable that contains the heat accumulation (e.g "GDH").

`Start_JDay` numeric parameter indicating the day when chill accumulation is supposed to start

`infocol` a vector of length length(Chill_req) which contains additional information for each element of the vector. This is preserved and included in the output. This only works when permutation=FALSE, and is meant to facilitate recognition of particular phenological events in the output.

Details

This function is an update to the bloom_prediction function, which was quite slow and didn't allow testing multiple chilling and heat requirements. In this updated version, chilling and heat requirements can be supplied as vectors, which are interpreted in sequence, with each pair of Chill_req and Heat_req values matched according to their position in the vectors. Through the permutations argument, it is also possible to compute stage occurrence dates for all possible combinations of the requirements specified by the Chill_req and Heat_req vectors.

The model allows specifying any numeric column as the chill and heat columns, indicated by the Chill_model and Heat_model parameters.

Value

data frame containing the predicted Julian dates of chilling requirement fulfillment and timing of the phenological stage. Columns are Season, Creq, Hreq, Creq_full (day when the chilling requirement is fulfilled) and Pheno_date (the predicted date of the phenological event).

Author(s)

Eike Luedeling

References

Model references:

Dynamic Model:

Erez A, Fishman S, Linsley-Noakes GC, Allan P (1990) The dynamic model for rest completion in peach buds. Acta Hortic 276, 165-174

Fishman S, Erez A, Couvillon GA (1987a) The temperature dependence of dormancy breaking in plants - computer simulation of processes studied under controlled temperatures. J Theor Biol 126(3), 309-321

Fishman S, Erez A, Couvillon GA (1987b) The temperature dependence of dormancy breaking in plants - mathematical analysis of a two-step model involving a cooperative transition. J Theor Biol 124(4), 473-483

Growing Degree Hours:

Anderson JL, Richardson EA, Kesner CD (1986) Validation of chill unit and flower bud phenology models for 'Montmorency' sour cherry. Acta Hortic 184, 71-78

Examples

```
hourtemps <- stack_hourly_temps(fix_weather(KA_weather[which(KA_weather$Year > 2008), ]),
                                latitude = 50.4)

CT <- chilling_hourtable(hourtemps, Start_JDay = 305)

bloom_prediction2(CT, c(30, 40, 50), c(1000, 1500, 2000))
bloom_prediction2(CT, c(30, 40, 50), c(1000, 1500, 2000), permutations = TRUE)
```

`bloom_prediction3` *Bloom prediction from chilling and forcing requirements, assumed to be fulfilled strictly in sequence - version 3*

Description

This is a pretty rudimentary function to predict phenological dates from chilling and forcing requirements and hourly chilling and forcing data. Note that there are enormous uncertainties in these predictions, which are hardly ever acknowledged. So please use this function with caution.

Usage

```
bloom_prediction3(
  hourtemps,
  Chill_req,
  Heat_req,
  models = c(Chill_Portions = Dynamic_Model, GDH = GDH_model),
  permutations = FALSE,
  Chill_model = "Chill_Portions",
  Heat_model = "GDH",
  Start_JDay = 305,
  infocol = NULL
)
```

Arguments

`hourtemps`	a data frame of hourly temperatures (e.g. resulting from the stack_hourly_temps function - should have columns "Year", "Month", "Day" and "Temp").
`Chill_req`	numeric vector indicating one or multiple chilling requirements of the particular growth stage (in the unit specified by "Chill_model")
`Heat_req`	numeric vector indicating one or multiple heat requirements of the particular growth stage (in Growing Degree Hours)
`models`	named list of models that should be applied to the hourly temperature data. These should be functions that take as input a vector of hourly temperatures. This defaults to c(Chill_Portions = Dynamic_Model, GDH = GDH_model), which refer to the Dynamic chill model and the Growing Degree Hours model functions contained in chillR.

permutations	boolean parameter indicating whether all possible combinations of the supplied chilling and heat requirements should be used. Defaults to FALSE, which means that the function matches chilling and heat requirements according to their positions in the Chill_req and Heat_req vectors and only predicts stage occurrence dates for these combinations.
Chill_model	character string specifying the chill model to use. This has to correspond to the name of the column in HourChillTable that contains the chill accumulation (default is "Chill_Portions" for units of the Dynamic Model).
Heat_model	character string specifying the heat model to use. This has to correspond to the name of the column in HourChillTable that contains the heat accumulation (e.g "GDH").
Start_JDay	numeric parameter indicating the day when chill accumulation is supposed to start. Note that this is also the latest acceptable bloom date.
infocol	a vector of length length(Chill_req) which contains additional information for each element of the vector. This is preserved and included in the output. This only works when permutation=FALSE, and is meant to facilitate recognition of particular phenological events in the output.

Details

This function is an update to the bloom_prediction and bloom_prediction2 functions. This version takes hourly temperatures as input rather than requiring pre-calculated chill and heat records. This functionality is now integrated in the function, so that users can now specify a list of temperature metrics/models to be computed and used in the bloom prediction.

Value

data frame containing the predicted Julian dates of chilling requirement fulfillment and timing of the phenological stage. Columns are Season, Creq, Hreq, Creq_full (day when the chilling requirement is fulfilled) and Pheno_date (the predicted date of the phenological event).

Author(s)

Eike Luedeling

References

Model references:

Dynamic Model:

Erez A, Fishman S, Linsley-Noakes GC, Allan P (1990) The dynamic model for rest completion in peach buds. Acta Hortic 276, 165-174

Fishman S, Erez A, Couvillon GA (1987a) The temperature dependence of dormancy breaking in plants - computer simulation of processes studied under controlled temperatures. J Theor Biol 126(3), 309-321

Fishman S, Erez A, Couvillon GA (1987b) The temperature dependence of dormancy breaking in plants - mathematical analysis of a two-step model involving a cooperative transition. J Theor Biol 124(4), 473-483

Growing Degree Hours:

Anderson JL, Richardson EA, Kesner CD (1986) Validation of chill unit and flower bud phenology models for 'Montmorency' sour cherry. Acta Hortic 184, 71-78

Examples

```
hourtemps <- stack_hourly_temps(fix_weather(KA_weather[which(KA_weather$Year > 2007), ]),
                                latitude = 50.4)

bloom_prediction3(hourtemps, c(30, 140, 50), c(1000, 1500, 2000))

bloom_prediction3(hourtemps, c(30, 40, 50), c(1000, 1500, 2000), permutations = TRUE,
                  Start_JDay = 1)

bloom_prediction3(hourtemps, c(300, 400, 600), c(100, 150, 200), permutations = TRUE,
                  Start_JDay = 1, models = c(CH = Chilling_Hours, Heat = GDD),
                  Chill_model = "CH", Heat_model = "Heat")
```

`bootstrap.phenologyFit`

bootstrap.phenologyFit

Description

This function bootstraps the residuals of a 'phenologyFit'. It internally calls 'phenologyFitter' on each bootstrap replicate.

Usage

```
bootstrap.phenologyFit(
  object,
  boot.R = 99,
 control = list(smooth = FALSE, verbose = FALSE, maxit = 1000, nb.stop.improvement =
    250),
  lower,
  upper,
  seed = 1766588
)
```

Arguments

`object`	class 'phenologyFit', the object to bootstrap
`boot.R`	integer. The number of bootstrap replicates
`control`	control parameters to 'GenSA', see 'GenSA::GenSA'
`lower`	Vector with length of 'par.guess'. Lower bounds for components.

upper	Vector with length of 'par.guess'. Upper bounds for components. If missing, 'upper' in 'object' is used.
seed	integer seed for the random number generator used by 'GenSA'. If missing, 'lower' in 'object' is used.

Details

bootstrap an object of S3 class 'phenologyFit'

Value

Invisibly returns a list with elements 'boot.R', 'object', 'seed', 'residuals', 'lower', 'upper', and 'res'. The latter list 'res' has 'boot.R' elements, which are lists again. Each of these lists contains named elements 'par', 'value', 'bloomJDays', and 'pbloomJDays'. 'par' are the best fit parameters on the particular bootstrap replicate, 'value' the corresponding RSS, 'bloomJDays' the re-sampled data and 'pbloomJDays' the predicted bloom JDays for this sample.

Author(s)

Carsten Urbach <urbach@hiskp.uni-bonn.de>

---

c.bootstrap_phenologyFit

Concatenate bootstrap_phenologyfit objects

Description

Concatenate bootstrap_phenologyfit objects

Usage

```
## S3 method for class 'bootstrap_phenologyFit'
c(...)
```

Arguments

...	Zero or multiple objects of type 'bootstrap_phenologyfit'.

Value

An object of class 'bootstrap_phenologyFit', the concatenation of the list of input object.

california_stations *Weather stations in California*

Description

This is a list of weather stations in California that are contained in the UC IPM database. This can also be generated with make_california_UCIPM_station_list(), but this takes quite a while. So this dataset is supposed to be a shortcut to this.

Format

a data.frame containing stations from the California UC IPM database (), with the columns: "Name", "Code", "Interval", "Lat", "Long", "Elev".

list("Name") name of the weather station

list("Code") code of the weather station, indicating the name and the database it comes from

list("Interval") period of available data (as character string)

list("Lat") latitude of the station

list("Long") longitude of the station

list("Elev") elevation of the station

Source

UC IPM website: http://www.ipm.ucdavis.edu/WEATHER/index.html

Examples

```
data(california_stations)
```

check_temperature_record *Check a daily or hourly temperature record for compliance with chillR's standards*

Description

This function performs basic tests to determine whether a temperature record complies with chillR's formatting rules. If desired, the function also checks whether the record is complete (has rows for all time units in the interval) and how many values are missing.

Usage

```
check_temperature_record(
  weather,
  hourly = FALSE,
  completeness_check = TRUE,
  no_variable_check = FALSE
)
```

Arguments

weather	object to be tested for whether it contains chillR-compatible temperature data.
hourly	boolean parameter indicating whether temp_record contains hourly data. If not, it is assumed to consist of daily records (the default).
completeness_check	boolean parameter indicating whether the records should be checked for completeness.
no_variable_check	boolean parameter to indicate whether the function should check if the dataset contains the usual chillR temperature variables. Defaults to TRUE, but should be set to FALSE for different data formats.

Value

list containing the following elements: 'data_frequency' ("daily or "hourly), 'weather_object' (boolean, indicates whether records are in a sub-object called weather), 'chillR_compliant' (boolean, indicates whether the object was found to conform to chillR format standards) and 'error' (contains error messages generated during the checking procedure).

Note

This function doesn't check whether there are faulty data. It only tests whether the data is compatible with the requirements of chillR's major functions.

Author(s)

Eike Luedeling

References

The chillR package:

Luedeling E, Kunz A and Blanke M, 2013. Identification of chilling and heat requirements of cherry trees - a statistical approach. International Journal of Biometeorology 57,679-689.

Examples

```
check_temperature_record(KA_weather)
```

`check_temperature_scenario` *Check temperature scenario for consistency*

Description

chillR's temperature generation procedures require absolute or relative temperature scenarios. This function checks these scenarios for consistency, regarding the data format, the reference year, and whether they are relative or absolute scenarios (based on specified criteria).

Usage

```
check_temperature_scenario(
  temperature_scenario,
  n_intervals = 12,
  check_scenario_type = TRUE,
  scenario_check_thresholds = c(-5, 10),
  update_scenario_type = TRUE,
  warn_me = TRUE,
  required_variables = c("Tmin", "Tmax")
)
```

Arguments

`temperature_scenario`
: can be one of two options: 1) a data.frame with two columns Tmin and Tmax and n_intervals (default: 12) rows containing temperature changes for all time intervals, or absolute temperatures for these intervals. 2) a temperature scenario object, consisting of the following elements: 'data' = a data frame with n_intervals elements containing the absolute or relative temperature information (as in input option 1); 'scenario_year' = the year the scenario is representative of; 'reference_year' = the year the scenario is representative of; 'scenario_type' = the scenario type ('absolute' or 'relative' - if NA, this is assigned automatically); 'labels' = and elements attached to the input temperature_scenario as an element names 'labels'. A subset of these elements can also be specified, but 'data' must be present.

`n_intervals`
: the number of time intervals specified in the temperature scenarios. This is often the number of months in a year, so the default is 12. If the temperature scenario is specified for a different number of time intervals, this should be adjusted.

`check_scenario_type`
: boolean variable indicating whether the specified (or unspecified) scenario type should be verified, i.e. whether the scenario is a relative or absolute temperature scenario.

`scenario_check_thresholds`
: vector with two numeric elements specifying the thresholds for checking whether the scenario is an absolute or relative temperature scenario. These are the minimum (first value) and maximum (second value) plausible changes in a relative

temperature scenario. The test only works in settings where either the lowest mean minimum temperature across all time intervals is below the stated minimum threshold or the highest mean maximum temperature across all time intervals is above the maximum threshold. With the default values c(-5,10), this should be the case for most locations on Earth, but in extreme cases (either for extreme change scenarios or where all monthly minimum and maximum temperatures are between -5 and 10 degrees), this may need adjustment. This is only used if check_scenario_type==TRUE.

`update_scenario_type`
: boolean variable stating whether, if scenario type is found to be inconsistent with the numbers, the scenario_type should be updated. Defaults to TRUE and is only used if check_scenario_type==TRUE.

`warn_me`
: boolean variable specifying whether warnings should be shown. Defaults to TRUE.

`required_variables`
: character vector containing the names of columns that are required. This defaults to c("Tmin","Tmax").

Details

Besides being able to validate classic temperature scenarios consisting of "Tmin" and "Tmax" data, the function can also validate other datasets (e.g. outputs of the getClimateWizardData function). To do this, the required variables should be provided as "required_variables" parameter. If there is no column "GCM" in the data element of the scenario, then the check_scenario_type parameter should be set to FALSE.

Value

temperature scenario object, consisting of the following elements: 'data' = a data frame with n_intervals elements containing the absolute or relative temperature information. 'reference_year' = the year the scenario is representative of. 'scenario_type' = the scenario type ('absolute' or 'relative'); 'labels' = and elements attached to the input temperature_scenario as an element names 'labels'.

The function also returns warnings, where elements are missing or the scenario_type appears to be wrong, and it stops with an error, if the scenario isn't specified in a format that is usable by chillR.

Author(s)

Eike Luedeling

Examples

```
temperature_scenario <- list(data = data.frame(Tmin = c(-5, -2, 0, 4, 9, 12,
                                                        15, 13, 12, 9, 4, 0),
                                               Tmax = c(0, 4, 8, 12, 15, 18,
                                                        21, 19, 17, 14, 11, 5)),
                             reference_year = 1975,
                             scenario_type = "absolute",
                             labels = list(GCM = "none",
```

```
                                        RCM = "none",
                                        Time = "1950-2000"))

checked_temperature_scenario <-
        check_temperature_scenario(temperature_scenario,
                                   n_intervals = 12,
                                   check_scenario_type = FALSE,
                                   scenario_check_thresholds = c(-5, 10),
                                   update_scenario_type = FALSE)

checked_temperature_scenario <-
        check_temperature_scenario(temperature_scenario,
                                   n_intervals = 12,
                                   check_scenario_type = TRUE,
                                   scenario_check_thresholds = c(-5, 10),
                                   update_scenario_type = FALSE)

checked_temperature_scenario <-
        check_temperature_scenario(temperature_scenario,
                                   n_intervals = 12,
                                   check_scenario_type = TRUE,
                                   scenario_check_thresholds = c(-5, 10),
                                   update_scenario_type = TRUE)
```

`chifull` *chifull*

Description

RSS to minimise by ‘phenologyFitter‘

Usage

```
chifull(par, modelfn, bloomJDays, SeasonList, na_penalty = 365, ...)
```

Arguments

`par`	numeric. vector of fit parameters
`modelfn`	function. model function
`bloomJDays`	numeric. vector of bloom hours! per year
`SeasonList`	list. list of index vectors per year.
`na_penalty`	numeric. penalty value for the residual if the model returns ‘NA‘.
`...`	further parameters to pass on to ‘modelfn‘.

Details

function to compute the RSS

`chile_agromet2chillR` *Convert a weather file downloaded from the Chilean Agromet website to chillR format*

Description

Convert downloaded weather data into a data frame that makes running other chillR functions easy.

Usage

```
chile_agromet2chillR(downloaded_weather_file, drop_most = TRUE)
```

Arguments

`downloaded_weather_file`	full path of a weather file downloaded from the Chilean Agromet website as an alleged Excel file (it has some formatting issues).
`drop_most`	boolean variable indicating if most columns should be dropped from the file. If set to TRUE (default), only essential columns for running chillR functions are retained.

Details

Processing the data with this function will make the data work well with the remainder of this package.

Value

a data.frame with weather data, according to the downloaded file provided as input. If drop_most is FALSE, all columns from the original dataset are preserved, although some column names are adjusted to chillR's preferences ("Year","Month","Day","Tmin","Tmax","Tmean","Prec", if these columns are present). If drop_most is TRUE, only columns likely to be of interest to chillR users are retained.

Note

Many databases have data quality flags, which may sometimes indicate that data aren't reliable. These are not considered by this function!

Author(s)

Eike Luedeling

References

The chillR package:

Luedeling E, Kunz A and Blanke M, 2013. Identification of chilling and heat requirements of cherry trees - a statistical approach. International Journal of Biometeorology 57,679-689.

Examples

```
weather<-fix_weather(KA_weather[which(KA_weather$Year>2005),])  # this line is
#only here to make the example run, even without downloading a file

# FOLLOW THE INSTRUCTIONS IN THE LINE BELOW THIS; AND THEN  RUN THE LINE
# AFTER THAT (without the #)
# download an Excel file from the website and save it to disk (path: {X})
#weather<-fix_weather(chile_agromet2chillR({x}))

hourtemps<-stack_hourly_temps(weather, latitude=50.4)
chilling(hourtemps,305,60)
```

---

chilling *Calculation of chilling and heat from hourly temperature records*

Description

Function to calculate three common horticultural chill metrics and one heat metric from stacked hourly temperatures (produced by stack_hourly_temps). Metrics that are calculated are Chilling Hours, Chill Units according to the Utah Model, Chill Portions according to the Dynamic Model and Growing Degree Hours.

Usage

```
chilling(
  hourtemps = NULL,
  Start_JDay = 1,
  End_JDay = 366,
  THourly = NULL,
  misstolerance = 50
)
```

Arguments

hourtemps	a list of two elements, with element 'hourtemps' being a dataframe of hourly temperatures (e.g. produced by stack_hourly_temps). This data frame must have a column for Year, a column for JDay (Julian date, or day of the year), a column for Hour and a column for Temp (hourly temperature). The second (optional) element is QC, which is a data.frame indicating completeness of the dataset. This is automatically produced by stack_hourly_temps.
Start_JDay	the start date (in Julian date, or day of the year) of the period, for which chill and heat should be quantified.
End_JDay	the end date (in Julian date, or day of the year) of the period, for which chill and heat should be quantified.
THourly	the same as hourtemps. This argument is only retained for downward compatibility and can be ignored in most cases.

misstolerance maximum percentage of values for a given season that can be missing without the record being removed from the output. Defaults to 50.

Details

Chill metrics are calculated as given in the references below. Chilling Hours are all hours with temperatures between 0 and 7.2 degrees C. Units of the Utah Model are calculated as suggested by Richardson et al. (1974) (different weights for different temperature ranges, and negation of chilling by warm temperatures). Chill Portions are calculated according to Fishman et al. (1987a,b). More honestly, they are calculated according to an Excel sheet produced by Amnon Erez and colleagues, which converts the complex equations in the Fishman papers into relatively simple Excel functions. These were translated into R. References to papers that include the full functions are given below. Growing Degree Hours are calculated according to Anderson et al. (1986), using the default values they suggest.

Value

data frame showing chilling and heat totals for the respective periods for all seasons included in the temperature records. Columns are Season, End_year (the year when the period ended), Days (the duration of the period), Chilling_Hours, Utah_Model, Chill_portions and GDH. If the weather input consisted of a list with elements hourtemps and QC, the output also contains columns from QC that indicate the completeness of the weather record that the calculations are based on.

Note

After doing extensive model comparisons, and reviewing a lot of relevant literature, I do not recommend using the Chilling Hours or Utah Models, especially in warm climates! The Dynamic Model (Chill Portions), though far from perfect, seems much more reliable.

Author(s)

Eike Luedeling

References

Model references:

Chilling Hours:

Weinberger JH (1950) Chilling requirements of peach varieties. Proc Am Soc Hortic Sci 56, 122-128

Bennett JP (1949) Temperature and bud rest period. Calif Agric 3 (11), 9+12

Utah Model:

Richardson EA, Seeley SD, Walker DR (1974) A model for estimating the completion of rest for Redhaven and Elberta peach trees. HortScience 9(4), 331-332

Dynamic Model:

Erez A, Fishman S, Linsley-Noakes GC, Allan P (1990) The dynamic model for rest completion in peach buds. Acta Hortic 276, 165-174

Fishman S, Erez A, Couvillon GA (1987a) The temperature dependence of dormancy breaking in plants - computer simulation of processes studied under controlled temperatures. J Theor Biol 126(3), 309-321

Fishman S, Erez A, Couvillon GA (1987b) The temperature dependence of dormancy breaking in plants - mathematical analysis of a two-step model involving a cooperative transition. J Theor Biol 124(4), 473-483

Growing Degree Hours:

Anderson JL, Richardson EA, Kesner CD (1986) Validation of chill unit and flower bud phenology models for 'Montmorency' sour cherry. Acta Hortic 184, 71-78

Model comparisons and model equations:

Luedeling E, Zhang M, Luedeling V and Girvetz EH, 2009. Sensitivity of winter chill models for fruit and nut trees to climatic changes expected in California's Central Valley. Agriculture, Ecosystems and Environment 133, 23-31

Luedeling E, Zhang M, McGranahan G and Leslie C, 2009. Validation of winter chill models using historic records of walnut phenology. Agricultural and Forest Meteorology 149, 1854-1864

Luedeling E and Brown PH, 2011. A global analysis of the comparability of winter chill models for fruit and nut trees. International Journal of Biometeorology 55, 411-421

Luedeling E, Kunz A and Blanke M, 2011. Mehr Chilling fuer Obstbaeume in waermeren Wintern? (More winter chill for fruit trees in warmer winters?). Erwerbs-Obstbau 53, 145-155

Review on chilling models in a climate change context:

Luedeling E, 2012. Climate change impacts on winter chill for temperate fruit and nut production: a review. Scientia Horticulturae 144, 218-229

The PLS method is described here:

Luedeling E and Gassner A, 2012. Partial Least Squares Regression for analyzing walnut phenology in California. Agricultural and Forest Meteorology 158, 43-52.

Wold S (1995) PLS for multivariate linear modeling. In: van der Waterbeemd H (ed) Chemometric methods in molecular design: methods and principles in medicinal chemistry, vol 2. Chemie, Weinheim, pp 195-218.

Wold S, Sjostrom M, Eriksson L (2001) PLS-regression: a basic tool of chemometrics. Chemometr Intell Lab 58(2), 109-130.

Mevik B-H, Wehrens R, Liland KH (2011) PLS: Partial Least Squares and Principal Component Regression. R package version 2.3-0. http://CRAN.R-project.org/package0pls.

Some applications of the PLS procedure:

Luedeling E, Kunz A and Blanke M, 2013. Identification of chilling and heat requirements of cherry trees - a statistical approach. International Journal of Biometeorology 57,679-689.

Yu H, Luedeling E and Xu J, 2010. Stronger winter than spring warming delays spring phenology on the Tibetan Plateau. Proceedings of the National Academy of Sciences (PNAS) 107 (51), 22151-22156.

Yu H, Xu J, Okuto E and Luedeling E, 2012. Seasonal Response of Grasslands to Climate Change on the Tibetan Plateau. PLoS ONE 7(11), e49230.

The exact procedure was used here:

Luedeling E, Guo L, Dai J, Leslie C, Blanke M, 2013. Differential responses of trees to temperature variation during the chilling and forcing phases. Agricultural and Forest Meteorology 181, 33-42.

The chillR package:

Luedeling E, Kunz A and Blanke M, 2013. Identification of chilling and heat requirements of cherry trees - a statistical approach. International Journal of Biometeorology 57,679-689.

Examples

```
# weather <- fix_weather(KA_weather[which(KA_weather$Year > 2006), ])
# hourtemps <- stack_hourly_temps(weather, latitude = 50.4)
# chilling(hourtemps, 305, 60)

chilling(stack_hourly_temps(fix_weather(KA_weather[which(KA_weather$Year > 2006), ]),
         latitude = 50.4))
```

Chilling_Hours *Calculation of cumulative chill according to the Chilling Hours Model*

Description

This function calculates winter chill for temperate trees according to the Chilling Hours Model.

Usage

```
Chilling_Hours(HourTemp, summ = TRUE)
```

Arguments

HourTemp	Vector of hourly temperatures.
summ	Boolean parameter indicating whether calculated metrics should be provided as cumulative values over the entire record (TRUE) or as the actual accumulation for each hour (FALSE).

Details

Chilling Hours are calculated as suggested by Bennett (1949) (all hours with temperatures between 0 and 7.2 degrees C are considered as one Chilling Hour.

Value

Vector of length length(HourTemp) containing the cumulative Chilling Hours over the entire duration of HourTemp.

Note

After doing extensive model comparisons, and reviewing a lot of relevant literature, I do not recommend using the Chilling Hours, especially in warm climates! The Dynamic Model (Chill Portions), though far from perfect, seems much more reliable.

Author(s)

Eike Luedeling

References

Chilling Hours references:

Weinberger JH (1950) Chilling requirements of peach varieties. Proc Am Soc Hortic Sci 56, 122-128

Bennett JP (1949) Temperature and bud rest period. Calif Agric 3 (11), 9+12

Examples

```
weather<-fix_weather(KA_weather[which(KA_weather$Year>2006),])

hourtemps<-stack_hourly_temps(weather,latitude=50.4)

Chilling_Hours(hourtemps$hourtemps$Temp)
```

---

chilling_hourtable *Add chilling and heat accumulation to table of hourly temperatures*

Description

This function calculates cumulative values for three chill metrics and one heat metric for every hour of an hourly temperature record. The count is restarted on a specified date each year.

Usage

```
chilling_hourtable(hourtemps, Start_JDay)
```

Arguments

hourtemps	a dataframe of stacked hourly temperatures (e.g. produced by stack_hourly_temps). This data frame must have a column for Year, a column for JDay (Julian date, or day of the year), a column for Hour and a column for Temp (hourly temperature).
Start_JDay	the start date (in Julian date, or day of the year) of the calculation for the four metrics. The count is restarted on this date every year.

Details

Chill metrics are calculated as given in the references below. Chilling Hours are all hours with temperatures between 0 and 7.2 degrees C. Units of the Utah Model are calculated as suggested by Richardson et al. (1974) (different weights for different temperature ranges, and negation of chilling by warm temperatures). Chill Portions are calculated according to Fishman et al. (1987a,b). More honestly, they are calculated according to an Excel sheet produced by Amnon Erez and colleagues, which converts the complex equations in the Fishman papers into relatively simple Excel functions. These were translated into R. References to papers that include the full functions are given below. Growing Degree Hours are calculated according to Anderson et al. (1986), using the default values they suggest.

Value

data frame consisting of all the columns of the THourly input data frame, plus the following additional columns: Chilling_Hours (cumulative number of Chilling Hours since the last Start_JDay), Chill_Portions (same for units of the Dynamic Models), Chill_Units (same for units of the Utah Model) and GDH (same for Growing Degree Hours).

Note

After doing extensive model comparisons, and reviewing a lot of relevant literature, I do not recommend using the Chilling Hours or Utah Models, especially in warm climates! The Dynamic Model (Chill Portions), though far from perfect, seems much more reliable.

Author(s)

Eike Luedeling

References

Model references:

Chilling Hours:

Weinberger JH (1950) Chilling requirements of peach varieties. Proc Am Soc Hortic Sci 56, 122-128

Bennett JP (1949) Temperature and bud rest period. Calif Agric 3 (11), 9+12

Utah Model:

Richardson EA, Seeley SD, Walker DR (1974) A model for estimating the completion of rest for Redhaven and Elberta peach trees. HortScience 9(4), 331-332

Dynamic Model:

Erez A, Fishman S, Linsley-Noakes GC, Allan P (1990) The dynamic model for rest completion in peach buds. Acta Hortic 276, 165-174

Fishman S, Erez A, Couvillon GA (1987a) The temperature dependence of dormancy breaking in plants - computer simulation of processes studied under controlled temperatures. J Theor Biol 126(3), 309-321

Fishman S, Erez A, Couvillon GA (1987b) The temperature dependence of dormancy breaking in plants - mathematical analysis of a two-step model involving a cooperative transition. J Theor Biol 124(4), 473-483

Growing Degree Hours:

Anderson JL, Richardson EA, Kesner CD (1986) Validation of chill unit and flower bud phenology models for 'Montmorency' sour cherry. Acta Hortic 184, 71-78

Model comparisons and model equations:

Luedeling E, Zhang M, Luedeling V and Girvetz EH, 2009. Sensitivity of winter chill models for fruit and nut trees to climatic changes expected in California's Central Valley. Agriculture, Ecosystems and Environment 133, 23-31

Luedeling E, Zhang M, McGranahan G and Leslie C, 2009. Validation of winter chill models using historic records of walnut phenology. Agricultural and Forest Meteorology 149, 1854-1864

Luedeling E and Brown PH, 2011. A global analysis of the comparability of winter chill models for fruit and nut trees. International Journal of Biometeorology 55, 411-421

Luedeling E, Kunz A and Blanke M, 2011. Mehr Chilling fuer Obstbaeume in waermeren Wintern? (More winter chill for fruit trees in warmer winters?). Erwerbs-Obstbau 53, 145-155

Review on chilling models in a climate change context:

Luedeling E, 2012. Climate change impacts on winter chill for temperate fruit and nut production: a review. Scientia Horticulturae 144, 218-229

The PLS method is described here:

Luedeling E and Gassner A, 2012. Partial Least Squares Regression for analyzing walnut phenology in California. Agricultural and Forest Meteorology 158, 43-52.

Wold S (1995) PLS for multivariate linear modeling. In: van der Waterbeemd H (ed) Chemometric methods in molecular design: methods and principles in medicinal chemistry, vol 2. Chemie, Weinheim, pp 195-218.

Wold S, Sjostrom M, Eriksson L (2001) PLS-regression: a basic tool of chemometrics. Chemometr Intell Lab 58(2), 109-130.

Mevik B-H, Wehrens R, Liland KH (2011) PLS: Partial Least Squares and Principal Component Regression. R package version 2.3-0. http://CRAN.R-project.org/package0pls.

Some applications of the PLS procedure:

Luedeling E, Kunz A and Blanke M, 2013. Identification of chilling and heat requirements of cherry trees - a statistical approach. International Journal of Biometeorology 57,679-689.

Yu H, Luedeling E and Xu J, 2010. Stronger winter than spring warming delays spring phenology on the Tibetan Plateau. Proceedings of the National Academy of Sciences (PNAS) 107 (51), 22151-22156.

Yu H, Xu J, Okuto E and Luedeling E, 2012. Seasonal Response of Grasslands to Climate Change on the Tibetan Plateau. PLoS ONE 7(11), e49230.

The exact procedure was used here:

Luedeling E, Guo L, Dai J, Leslie C, Blanke M, 2013. Differential responses of trees to temperature variation during the chilling and forcing phases. Agricultural and Forest Meteorology 181, 33-42.

The chillR package:

Luedeling E, Kunz A and Blanke M, 2013. Identification of chilling and heat requirements of cherry trees - a statistical approach. International Journal of Biometeorology 57,679-689.

Examples

```
weather<-fix_weather(KA_weather[which(KA_weather$Year>2008),])

hourtemps<-stack_hourly_temps(weather,latitude=50.4)

cht<-chilling_hourtable(hourtemps,20)
```

ChuineCF *ChuineCF*

Description

chilling and forcing response function for the unified model by Chuine

Usage

```
ChuineCF(x, a, b, c)
```

Arguments

x	temperature
a	numeric. paramter
b	numeric. paramter
c	numeric. paramter

Value

Returns a numeric vector.

References

Isabelle Chuine, A Unified Model for Budburst of Trees, J. theor. Biol. (2000) 207

ChuineFstar *ChuineFstar*

Description

Critical forcing value

Usage

```
ChuineFstar(Ctot, w, k)
```

Arguments

Ctot	numeric. total state of chilling
w	numeric > 0. parameter.
k	numeric < 0. parameter.

Value

Returns a numeric vector.

References

Isabelle Chuine, A Unified Model for Budburst of Trees, J. theor. Biol. (2000) 207

---

color_bar_maker *Make color scheme for bar plots in outputs of the chillR package*

Description

Function to make color schemes for color bar plots in the chillR package. Colors are assigned based on values in two columns of a data frame. One column contains a threshold, below which col3 is assigned. If values are above the threshold, the value in the other column determines the color: col1 if the value is negative, col2 if positive. This function is useful for making the PLS output figures in the chillR package.

Usage

```
color_bar_maker(column_yn, column_quant, threshold, col1, col2, col3)
```

Arguments

column_yn	numeric vector containing the data, on which the threshold is to be applied. In the case of the PLS output, this is the data from the VIP column.
column_quant	numeric vector containing the data that determines whether items from column_yn that are above the threshold get assigned col1 or col2.
threshold	threshold for values from column_yn to be used for deciding which bars should get col3 and which ones should move on to the next decision step (col1 or col2)
col1	a color (either a color name or a number) this is applied where column_yn is above the threshold, and column_quant is negative
col2	a color (either a color name or a number) this is applied where column_yn is above the threshold, and column_quant is positive
col3	a color (either a color name or a number) this is applied where column_yn is below the threshold

Value

a vector of colors, which can be used as col argument when making plots

Author(s)

Eike Luedeling

References

Luedeling E, Kunz A and Blanke M, 2013. Identification of chilling and heat requirements of cherry trees - a statistical approach. International Journal of Biometeorology 57,679-689.

Examples

```
PLS_results<-PLS_pheno(
  weather_data=make_all_day_table(KA_weather),
  split_month=6,   #last month in same year
  bio_data=KA_bloom)

colbar<-color_bar_maker(PLS_results$PLS_summary$VIP,PLS_results$PLS_summary$Coef,0.8,
       "RED","DARK GREEN","GREY")
```

---

convert_scen_information

Converts list of change scenarios to data.frame or vice versa

---

Description

Allows the user to convert a list of change scenarios to a single data.frame or vice versa. If it converts from data.frame to list, the user can decide if the returned list should be flat or structured. In case of a list of change scenarios, the list should have named elements. In case of composite names, the function assumes that the location is the first part of the composite name, composite elements are seperated by dot.

Usage

```
convert_scen_information(scenario_object, give_structure = TRUE)
```

Arguments

scenario_object	can be either a data.frame or a list of change scenarios. If it is a data.frame, it containing the relative change scenarios
give_structure	boolean, by default set TRUE. If set TRUE, then the output is a nested list of the structure: 1) Location 2)SSP 3)GCM 4)Timepoint. If set FALSE, then returns flat list with names following the scheme: Location.SSP.GCM.Timepoint.

Value

list / data.frame with relative change scenarios

Author(s)

Lars Caspersen

Examples

```
## Not run:
download_cmip6_ecmwfr(scenario = 'ssp1_2_6',
                      area = c(55, 5.5, 47, 15.1),
                      user = 'write user id here',
                      key = 'write key here',
                      model = 'AWI-CM-1-1-MR',
                      frequency = 'monthly',
                      variable = c('Tmin', 'Tmax'),
                      year_start = 2015,
                      year_end = 2100)

download_baseline_cmip6_ecmwfr(
    area = c(55, 5.5, 47, 15.1),
    user = 'write user id here',
    key = 'write key here',
    model = 'AWI-CM-1-1-MR',
    frequency = 'monthly',

station <- data.frame(
      station_name = c('Zaragoza', 'Klein-Altendorf', 'Sfax',
      'Cieza', 'Meknes', 'Santomera'),
      longitude = c(-0.88,  6.99, 10.75, -1.41, -5.54, -1.05),
      latitude = c(41.65, 50.61, 34.75, 38.24, 33.88, 38.06))

extracted <- extract_cmip6_data(stations = station)

scenario_df <- gen_rel_change_scenario(extracted)

scenario_list <- convert_scen_information(scenario_df)


## End(Not run)
```

`daily_chill` *Calculation of daily chill and heat accumulation*

Description

This function calculates daily chill (with three models) and heat accumulation for every day of an hourly temperature record (best generated with stack_hourly_temps). It includes the option to include calculation of a running mean, which smoothes accumulation curves. Especially for the Dynamic Model, this may be advisable, because it does not accumulate chill smoothly, but rather in steps.

Usage

```
daily_chill(
  hourtemps = NULL,
  running_mean = 1,
  models = list(Chilling_Hours = Chilling_Hours, Utah_Chill_Units = Utah_Model,
    Chill_Portions = Dynamic_Model, GDH = GDH),
  THourly = NULL
)
```

Arguments

hourtemps	a dataframe of stacked hourly temperatures (e.g. produced by stack_hourly_temps). This data frame must have a column for Year, a column for JDay (Julian date, or day of the year), a column for Hour and a column for Temp (hourly temperature).
running_mean	what running mean should be applied to smooth the chill and heat accumulation curves? This should be an odd integer. Use 1 (default) for no smoothing.
models	named list of models that should be applied to the hourly temperature data. These should be functions that take as input a vector of hourly temperatures. This defaults to the set of models provided by the chilling function.
THourly	hourtemps was called THourly in an earlier version of this package. So in order to allow function calls written before the 0.57 update to still work, this is included here.

Details

Temperature metrics are calculated according to the specified models. They are computed based on hourly temperature records and then summed to produce daily chill accumulation rates.

Value

a daily chill object consisting of the following elements

object_type	a character string "daily_chill" indicating that this is a daily_chill object
daily_chill	data frame consisting of the columns YYMMDD, Year, Month, Day and Tmean, plus one column for each model that is evaluated. The latter columns have the name given to the model in the models list and they contain daily total accumulations of the computed metrics.

Note

After doing extensive model comparisons, and reviewing a lot of relevant literature, I do not recommend using the Chilling Hours or Utah Models, especially in warm climates! The Dynamic Model (Chill Portions), though far from perfect, seems much more reliable.

Author(s)

Eike Luedeling

References

Model references for the default models:

Chilling Hours:

Weinberger JH (1950) Chilling requirements of peach varieties. Proc Am Soc Hortic Sci 56, 122-128

Bennett JP (1949) Temperature and bud rest period. Calif Agric 3 (11), 9+12

Utah Model:

Richardson EA, Seeley SD, Walker DR (1974) A model for estimating the completion of rest for Redhaven and Elberta peach trees. HortScience 9(4), 331-332

Dynamic Model:

Erez A, Fishman S, Linsley-Noakes GC, Allan P (1990) The dynamic model for rest completion in peach buds. Acta Hortic 276, 165-174

Fishman S, Erez A, Couvillon GA (1987a) The temperature dependence of dormancy breaking in plants - computer simulation of processes studied under controlled temperatures. J Theor Biol 126(3), 309-321

Fishman S, Erez A, Couvillon GA (1987b) The temperature dependence of dormancy breaking in plants - mathematical analysis of a two-step model involving a cooperative transition. J Theor Biol 124(4), 473-483

Growing Degree Hours:

Anderson JL, Richardson EA, Kesner CD (1986) Validation of chill unit and flower bud phenology models for 'Montmorency' sour cherry. Acta Hortic 184, 71-78

Model comparisons and model equations:

Luedeling E, Zhang M, Luedeling V and Girvetz EH, 2009. Sensitivity of winter chill models for fruit and nut trees to climatic changes expected in California's Central Valley. Agriculture, Ecosystems and Environment 133, 23-31

Luedeling E, Zhang M, McGranahan G and Leslie C, 2009. Validation of winter chill models using historic records of walnut phenology. Agricultural and Forest Meteorology 149, 1854-1864

Luedeling E and Brown PH, 2011. A global analysis of the comparability of winter chill models for fruit and nut trees. International Journal of Biometeorology 55, 411-421

Luedeling E, Kunz A and Blanke M, 2011. Mehr Chilling fuer Obstbaeume in waermeren Wintern? (More winter chill for fruit trees in warmer winters?). Erwerbs-Obstbau 53, 145-155

Review on chilling models in a climate change context:

Luedeling E, 2012. Climate change impacts on winter chill for temperate fruit and nut production: a review. Scientia Horticulturae 144, 218-229

The PLS method is described here:

Luedeling E and Gassner A, 2012. Partial Least Squares Regression for analyzing walnut phenology in California. Agricultural and Forest Meteorology 158, 43-52.

Wold S (1995) PLS for multivariate linear modeling. In: van der Waterbeemd H (ed) Chemometric methods in molecular design: methods and principles in medicinal chemistry, vol 2. Chemie, Weinheim, pp 195-218.

Wold S, Sjostrom M, Eriksson L (2001) PLS-regression: a basic tool of chemometrics. Chemometr Intell Lab 58(2), 109-130.

Mevik B-H, Wehrens R, Liland KH (2011) PLS: Partial Least Squares and Principal Component Regression. R package version 2.3-0. http://CRAN.R-project.org/package0pls.

Some applications of the PLS procedure:

Luedeling E, Kunz A and Blanke M, 2013. Identification of chilling and heat requirements of cherry trees - a statistical approach. International Journal of Biometeorology 57,679-689.

Yu H, Luedeling E and Xu J, 2010. Stronger winter than spring warming delays spring phenology on the Tibetan Plateau. Proceedings of the National Academy of Sciences (PNAS) 107 (51), 22151-22156.

Yu H, Xu J, Okuto E and Luedeling E, 2012. Seasonal Response of Grasslands to Climate Change on the Tibetan Plateau. PLoS ONE 7(11), e49230.

The exact procedure was used here:

Luedeling E, Guo L, Dai J, Leslie C, Blanke M, 2013. Differential responses of trees to temperature variation during the chilling and forcing phases. Agricultural and Forest Meteorology 181, 33-42.

The chillR package:

Luedeling E, Kunz A and Blanke M, 2013. Identification of chilling and heat requirements of cherry trees - a statistical approach. International Journal of Biometeorology 57,679-689.

Examples

```
models<-list(CP=Dynamic_Model,CU=Utah_Model,GDH=GDH)

dc<-daily_chill(stack_hourly_temps(fix_weather(KA_weather[which(KA_weather$Year>2009),]),
 latitude=50.4),11,models)
```

`Date2YEARMODA` *Date to YEARMODA conversion*

Description

Converts R dates to YEARMODA format

Usage

```
Date2YEARMODA(Date, hours = FALSE)
```

Arguments

`Date`	Date in R date format
`hours`	boolean variable indicating whether YEARMODAHO should be calculated (YEARMODA + hours)

Details

Converts R date to YEARMODA

Value

YEARMODA object (e.g. 20111224 for 24th December 2011)

Author(s)

Eike Luedeling

Examples

```
Date2YEARMODA(YEARMODA2Date(20001205))
Date2YEARMODA(YEARMODA2Date(19901003))
```

---

daylength *Compute sunrise and sunset times, and daylength*

Description

This function computes sunrise time, sunset time and daylength for a particular location and day of the year (Julian day). This is done using equations by Spencer (1971) and Almorox et al. (2005).

Usage

```
daylength(latitude, JDay, notimes.as.na = FALSE)
```

Arguments

latitude	numeric value specifying the geographic latitude (in decimal degrees) of the location of interest
JDay	numeric (usually integer) value or vector specifying the Julian day (day of the year), for which calculations should be done.
notimes.as.na	parameter to determine whether for days without sunrise or sunset, na should be returned for Sunset and Sunrise. If left at FALSE (the default), the function returns -99 and 99 for sunrise and sunset or polar nights and polar days, respectively.

Value

list with three elements Sunrise, Sunset and Daylength. For days without sunrise (polar nights), sunset and sunrise become -99 and the daylength 0. For days without sunset, sunset and sunrise are 99 and daylength 24.

Author(s)

Eike Luedeling

References

Spencer JW, 1971. Fourier series representation of the position of the Sun. Search 2(5), 172.

Almorox J, Hontoria C and Benito M, 2005. Statistical validation of daylength definitions for estimation of global solar radiation in Toledo, Spain. Energy Conversion and Management 46(9-10), 1465-1471)

Examples

```
daylength(latitude=50,JDay=40)
plot(daylength(latitude=35,JDay=1:365)$Daylength)
```

---

download_baseline_cmip6_ecmwfr

Download historical CMIP6 Data via the ecwfr package

Description

Accesses the CMIP6 data of the Copernicus API via the ecmwfr package. Saves the downloaded files as .zip objects in the specified path in a subfolder with the coordinates of the downloaded area as subfolder name. You can either specify the GCMs by name or you can take all GCMs for which you downloaded climate change scenarios (model = "match_downloaded").

Usage

```
download_baseline_cmip6_ecmwfr(
  area,
  model = "match_downloaded",
  service = "cds",
  frequency = "monthly",
  variable = c("Tmin", "Tmax"),
  year_start = 1986,
  year_end = 2014,
  month = 1:12,
  sec_wait = 3600,
  n_try = 10,
  update_everything = FALSE,
  path_download = "cmip6_downloaded",
  user = "ecmwfr",
  key = NULL
)
```

Arguments

area	numeric vector of length 4. Sets the spatial boundaries of the downloaded data. Coordinates are supplied in the following format: c(maximum latitude, minimum longitude, minimum latitude, maximum longitude), which corresponds to the northern extent, western extent, southern extent and eastern extent of the area of interest.
model	character, by default "match_downloaded". Looks up the already downloaded GCMs for the climate change scenarios of the "download_cmip6_ecmwfr()" function. You can also specify the models by name as a vector.
service	character, by default 'cds'. Decides which database is used. For more details see in the documentation of ecmwfr::wf_set_key().
frequency	character, can be either 'daily' or 'monthly'. Sets if the downloaded CMIP6 data is in daily or monthly format.
variable	vector of characters, decides which variables get downloaded. Currently, the options "Tmin" (Daily minimum temperature in degree centigrade), "Tmax" (Daily maximum temperature in degree centigrade) and "Prec" (Daily sum of precipitation in mm) are the only valid options.
year_start	numeric, earliest year for downloaded CMIP6 data. By default set to 1985.
year_end	numeric, latest year for downloaded CMIP6 data. By default set to 2014.
month	numeric vector, sets for which months data should be downloaded. By default set to 1:12.
sec_wait	numeric, sets the maximum waiting time per requested file. By default is 3600, so 1 hour.
n_try	numeric, number of repeated calls for the API. For more information see 'Details'.
update_everything	logical, by default set to FALSE. When set to FALSE, scenarios with matching names that have already been downloaded are skipped. If set to TRUE, then files are downloaded regardless if a file with the same name is already present.
path_download	character, sets the path for the download of the CMIP6 file. If not already present, then a new folder will be created. The path is relative to the working directory.
user	a character, user name provided by ECMWF data service. The default "ecmwfr" should be fine. Otherwise provide the email address which was used to sign-up at ECMWF / Copernicus Climate Data Store
key	a character. Can be found just beneath the user id on the profile when registering for the Copernicus website next to "Personal Access Token". Should be provided as a character (so in quotation marks).

Details

Registering for cds.climate.coperincus.eu: point to the registration link in the top right corner of https://cds.climate.copernicus.eu/datasets

Value

NULL, the downloaded files are saved in the stated directory

Author(s)

Lars Caspersen

Examples

```
## Not run:
# example with one specified GCM
download_baseline_cmip6_ecmwfr(
    area = c(55, 5.5, 47, 15.1),
    model = 'AWI-CM-1-1-MR',
    frequency = 'monthly',
    variable = c('Tmin', 'Tmax'))

## End(Not run)
```

---

download_cmip6_ecmwfr *Download CMIP6 Data via the ecwfr package*

Description

Accesses the CMIP6 data of the Copernicus API via the ecmwfr package. Saves the downloaded files as .zip objects in the specified path in a subfolder with the coordinates of the downloaded area as subfolder name. You can either specify the GCMs by name, take the combinations of scenario and GCM that worked in the past (model = 'default') or you can try out all GCMs for a scenario and take the ones for which there is data (model = 'all').

Usage

```
download_cmip6_ecmwfr(
  scenarios,
  area,
  model = "default",
  service = "cds",
  frequency = "monthly",
  variable = c("Tmin", "Tmax"),
  year_start = 2015,
  year_end = 2100,
  month = 1:12,
  sec_wait = 3600,
  n_try = 10,
  update_everything = FALSE,
  path_download = "cmip6_downloaded",
  user = "ecmwfr",
```

```
  key = NULL
)
```

Arguments

scenarios	vector of characters specifying the shared socioeconomic pathway scenarios (SSP) to be downloaded. Currently the values 'ssp126', 'ssp245', 'ssp370' and 'ssp585' are the only accepted options. These are the standard scenarios of CMIP6.
area	numeric vector of length 4. Sets the spatial boundaries of the downloaded data. Coordinates are supplied in the following format: c(maximum latitude, minimum longitude, minimum latitude, maximum longitude), which corresponds to the northern extent, western extent, southern extent and eastern extent of the area of interest.
model	character, by default "default". Decides which global climate models are requested. If set to "default" then depending on the scenario and temporal resolution around 20 models are selected for which we know that certain combinations of scenario and variables are available. If this is set to "all", then all potential models are requested. You can also hand-pick the models you want to download as a vector of the model names. You can check https://cds.climate.copernicus.eu/datasets/projections-cmip6?tab=download for the list of models. In case a certain request fails because either the model name is wrong or the requested combination of SSP, time period and variable is not available, then the model is dropped from the requests and the function carries on with the remaining requests. The user will get a warning in these cases.
service	character, by default 'cds'. Decides which database is used. For more details see in the documentation of ecmwfr::wf_set_key().
frequency	character, can be either 'daily' or 'monthly'. Sets if the downloaded CMIP6 data is in daily or monthly format.
variable	vector of characters, decides which variables get downloaded. Currently, the options "Tmin" (Daily minimum temperature in degree centigrade), "Tmax" (Daily maximum temperature in degree centigrade) and "Prec" (Daily sum of precipitation in mm) are the only valid options.
year_start	numeric, earliest year for downloaded CMIP6 data. By default set to 2015.
year_end	numeric, latest year for downloaded CMIP6 data. By default set to 2100.
month	numeric vector, sets for which months data should be downloaded. By default set to 1:12.
sec_wait	numeric, sets the maximum waiting time per requested file. By default is 3600, so 1 hour.
n_try	numeric, number of repeated calls for the API. For more information see 'Details'.
update_everything	logical, by default set to FALSE. When set to FALSE, scenarios with matching names that have already been downloaded are skipped. If set to TRUE, then files are downloaded regardless if a file with the same name is already present.

`path_download` character, sets the path for the download of the CMIP6 file. If not already present, then a new folder will be created. The path is relative to the working directory.

`user` a character, user name provided by ECMWF data service. The default "ecmwfr" should be fine. Otherwise provide the email address which was used to sign-up at ECMWF / Copernicus Climate Data Store

`key` a character. Can be found just beneath the user id on the profile when registering for the Copernicus website next to "Personal Access Token". Should be provided as a character (so in quotation marks).

Details

Registering for cds.climate.coperincus.eu: Point to the registration link in the top right corner of https://cds.climate.copernicus.eu/datasets

Finding the user id and the key:

On the website of the Copernicus climate data store, navigate to the user profile and scroll to the bottom to "API key". There you can find the item "UID". The user id should be provided as character (within quotation marks). Just below, you can also find the key, which is also needed when using this function.

After successful registration some extra steps are needed in order to be able to download CMIP6 data. In addition to the "Terms of use of the Copernicus Climate Store" and the "Data Protection and Privacy Agreement", you also need to agree to the "CMIP6 - Data Access - Terms of Use". This needs to be done after registering. You can agree to the terms via the following link and scroll to the bottom of the page: https://cds.climate.copernicus.eu/datasets/projections-cmip6?tab=download.

Alternatively, you can navigate to the terms within the Copernicus webpage. Go to "Datasets", you can find it in the upper ribbon of the main page. There you need to search for "CMIP6" using the search field and choose the first result, which is named "CMIP6 climate projections". There you need to click on "Download data" and scroll to the very bottom of the page to the field "Terms of Use". There you need to click on the button saying "Accept Terms". If you do not accept the terms the download via the API (and consequently via this function) will not be possible!

Sometimes the server is not responding in time, which can make the download fail. In such cases, after a short waiting time of 5 seconds, the request is started again. If the error reoccurs several times, the requested model will be dropped from the list of requests. By default the number of allowed repeated requests is 10. The user will get a warning if the model is dropped from the requests.

Value

NULL, the downloaded files are saved in the stated directory

Author(s)

Lars Caspersen, Antonio Picornell

Examples

```
## Not run:
# example with one specified GCM
download_cmip6_ecmwfr(
    scenarios = 'ssp126',
    area = c(55, 5.5, 47, 15.1),
    key = 'write key here',
    model = 'AWI-CM-1-1-MR',
    frequency = 'monthly',
    variable = c('Tmin', 'Tmax'),
    year_start = 2015,
    year_end = 2100)

# example with default combinations of scenario and GCM
download_cmip6_ecmwfr(
    scenarios = 'ssp126',
    area = c(55, 5.5, 47, 15.1),
    key = 'write key here',
    model = 'default',
    frequency = 'monthly',
    variable = c('Tmin', 'Tmax'),
    year_start = 2015,
    year_end = 2100)

# example with all possible combinations of scenario and GCM
# this may take a little longer
download_cmip6_ecmwfr(
    scenarios = 'ssp126',
    area = c(55, 5.5, 47, 15.1),
    key = 'write key here',
    model = 'all',
    frequency = 'monthly',
    variable = c('Tmin', 'Tmax'),
    year_start = 2015,
    year_end = 2100)

## End(Not run)
```

`Dynamic_Model` *Dynamic_Model*

Description

Calculation of cumulative chill according to the Dynamic Model

This function calculates winter chill for temperate trees according to the Dynamic Model.

Chill Portions are calculated as suggested by Erez et al. (1990).

Usage

```
Dynamic_Model(
  HourTemp,
  summ = TRUE,
  E0 = 4153.5,
  E1 = 12888.8,
  A0 = 139500,
  A1 = 2.567e+18,
  slope = 1.6,
  Tf = 277
)
```

Arguments

HourTemp	Vector of hourly temperatures in degree Celsius.
summ	Boolean parameter indicating whether calculated metrics should be provided as cumulative values over the entire record (TRUE) or as the actual accumulation for each hour (FALSE).
E0	numeric. Parameter E_0 of the dynamic model
E1	numeric. Parameter E_1 of the dynamic model
A0	numeric. Parameter A_0 of the dynamic model
A1	numeric. Parameter A_1 of the dynamic model
slope	numeric. Slope parameter for sigmoidal function
Tf	numeric. Transition temperature (in degree Kelvin) for the sigmoidal function.

Value

Vector of length length(HourTemp) containing the cumulative Chill Portions over the entire duration of HourTemp.

Author(s)

Eike Luedeling

References

Dynamic Model references:

Erez A, Fishman S, Linsley-Noakes GC, Allan P (1990) The dynamic model for rest completion in peach buds. Acta Hortic 276, 165-174

Fishman S, Erez A, Couvillon GA (1987a) The temperature dependence of dormancy breaking in plants - computer simulation of processes studied under controlled temperatures. J Theor Biol 126(3), 309-321

Fishman S, Erez A, Couvillon GA (1987b) The temperature dependence of dormancy breaking in plants - mathematical analysis of a two-step model involving a cooperative transition. J Theor Biol 124(4), 473-483

Examples

```
weather<-fix_weather(KA_weather[which(KA_weather$Year>2006),])

hourtemps<-stack_hourly_temps(weather,latitude=50.4)

res <- Dynamic_Model(hourtemps$hourtemps$Temp)
```

DynModel_driver *DynModel_driver*

Description

Calculation of cumulative chill according to the Dynamic Model

This function calculates winter chill for temperate trees according to the Dynamic Model.

Chill Portions are calculated as suggested by Erez et al. (1990).

Usage

```
DynModel_driver(
  temp,
  times,
  A0 = 139500,
  A1 = 2.567e+18,
  E0 = 4153.5,
  E1 = 12888.8,
  slope = 1.6,
  Tf = 4,
  deg_celsius = TRUE
)
```

Arguments

temp	Vector of temperatures.
times	numeric vector. Optional times at which the temperatures where measured, if not given, hourly temperatures will be assumed
A0	numeric. Parameter A_0 of the dynamic model
A1	numeric. Parameter A_1 of the dynamic model
E0	numeric. Parameter E_0 of the dynamic model
E1	numeric. Parameter E_1 of the dynamic model
slope	numeric. Slope parameter for sigmoidal function
Tf	numeric. Transition temperature (in degree Kelvin) for the sigmoidal function
deg_celsius	boolean. whether or not the temperature vector and the model temperature parameters are in degree Celsius (Kelvin otherwise)

Details

This function gives idential results as Dynamic_Model for hourly temperature data, returns more details but is also a bit slower in the R code version

Value

List containint four vectors of length(temp) with elements `x` is the PDBF, `y` the accumulated chill, `delta` the chill portions and `xs`, which is $x_s = A_0/A_1 \exp(-(E_0 - E_1)/T)$ Portions over the entire duration of HourTemp.

Author(s)

Carsten Urbach <urbach@hiskp.uni-bonn.de>

References

Dynamic Model references:

Erez A, Fishman S, Linsley-Noakes GC, Allan P (1990) The dynamic model for rest completion in peach buds. Acta Hortic 276, 165-174

Fishman S, Erez A, Couvillon GA (1987a) The temperature dependence of dormancy breaking in plants - computer simulation of processes studied under controlled temperatures. J Theor Biol 126(3), 309-321

Fishman S, Erez A, Couvillon GA (1987b) The temperature dependence of dormancy breaking in plants - mathematical analysis of a two-step model involving a cooperative transition. J Theor Biol 124(4), 473-483

Examples

```
weather<-fix_weather(KA_weather[which(KA_weather$Year>2006),])
hourtemps<-stack_hourly_temps(weather,latitude=50.4)
res2 <- DynModel_driver(temp=hourtemps$hourtemps$Temp)
```

Empirical_daily_temperature_curve

Empirical daily temperature curve

Description

This function derives an empirical daily temperature curve from observed hourly temperature data. The mean temperature during each hour of the day is expressed as a function of the daily minimum and maximum temperature. This is done separately for each month of the year. The output is a data.frame that can then be used with the `Empirical_hourly_temperatures` function to generate hourly temperatures from data on daily minimum (`Tmin`) and maximum (`Tmax`) temperatures.

Usage

```
Empirical_daily_temperature_curve(Thourly)
```

Arguments

`Thourly`	data.frame containing hourly temperatures. Must contain columns `Year` (year of observation), `Month` (month of observation), `Day` (day of observation), `Hour` (hour of observation) and `Temp` (Observed temperature). If multiple observations within an hour are available, these are averaged.

Value

data.frame containing three columns: `Month` (month for which coefficient applies), `Hour` (hour for which coefficient applies) and `Prediction_coefficient` (the coefficient used for empirical temperature prediction). Coefficients indicate, by what fraction of the daily temperature range the temperature during the specified hour is above the daily minimum temperature.

Author(s)

Eike Luedeling

Examples

```
Empirical_daily_temperature_curve(Winters_hours_gaps)
```

Empirical_hourly_temperatures

Empirical daily temperature prediction

Description

This function generates hourly temperatures from daily minimum and maximum temperatures, based on an empirical relationship of these two daily temperature extremes with the hourly temperature. Usually, this relationship will have been determined with the `Empirical_daily_temperature_curve` function.

Usage

```
Empirical_hourly_temperatures(Tdaily, empi_coeffs)
```

Arguments

Tdaily	data.frame containing daily minimum and maximum temperatures. Must contain columns Year (year of observation), Month (month of observation), Day (day of observation), Tmin (Minimum daily temperature) and Tmax (Maximum daily temperature).
empi_coeffs	data.frame containing coefficients for the hourly temperature prediction, e.g. generated with the function Empirical_daily_temperature_curve. Needs to contain the following columns: Month (month for which coefficient applies), Hour (hour for which coefficient applies) and Prediction_coefficient (the coefficient used for empirical temperature prediction). Coefficients indicate, by what fraction of the daily temperature range the temperature during the specified hour is above the daily minimum temperature.

Value

data.frame containing all columns of the Tdaily dataset, but also the columns Hour and Temp, for the hour of the day and the predicted temperature, respectively.

Author(s)

Eike Luedeling

Examples

```
coeffs<-Empirical_daily_temperature_curve(Winters_hours_gaps)
Winters_daily<-make_all_day_table(Winters_hours_gaps, input_timestep="hour")
Empirical_hourly_temperatures(Winters_daily,coeffs)
```

---

extract_cmip6_data *Unpacks and formats downloaded CMIP6 data*

Description

Opens the downloaded .zip files and returns the CMIP6 climate projections for specified locations .

Usage

```
extract_cmip6_data(
  stations,
  variable = c("Tmin", "Tmax"),
  download_path = "cmip6_downloaded",
  keep_downloaded = TRUE
)
```

Arguments

stations	data.frame with the locations of interest, for which the CMIP6 data should be extracted. Needs to contain the columns 'longitude', 'latitude' and 'station_name'.
variable	character, decides which variables from the downloaded files get read. Currently, valid options are "Tmin", "Tmax" and "Prec". The value is usually the same as in download_cmip6_ecmwfr function.
download_path	character, sets the path for the download of the CMIP6 file. If not already present, then a new folder will be created. The path is relative to working directory.
keep_downloaded	Boolean, by default set to TRUE. If TRUE, the function will not delete the downloaded .nc files. This makes sense when the user may want to use the climate change data for other locations.

Value

named list of data.frames. Element names follow the syntax 'SSP'_'GCM', where SSP is the shared socioeconomic pathway and GCM is the global climate model that generated the weather data. The data.frames contain the extracted values for the requested locations.

Author(s)

Lars Caspersen

Examples

```
## Not run:
scenario<-c("ssp126", "ssp245", "ssp370", "ssp585")

download_cmip6_ecmwfr(scenario,
                        key = 'your-key-here'
                        user = 'your-user-name-here',
                        area =  c(52, -7, 33, 8) )


station <- data.frame(
    station_name = c('Zaragoza', 'Klein-Altendorf', 'Sfax', 'Cieza',
        'Meknes', 'Santomera'),
    longitude = c(-0.88,  6.99, 10.75, -1.41, -5.54, -1.05),
    latitude = c(41.65, 50.61, 34.75, 38.24, 33.88, 38.06))

extracted <- extract_cmip6_data(
    stations = station)

scenario <- gen_rel_change_scenario(
    extracted, years_local_weather = c(1992, 2021))


## End(Not run)
```

extract_differences_between_characters

Identify shared leading or trailing character strings

Description

For a vector of character strings, identify elements between shared leading and/or trailing substrings, e.g. for a vector such as c("XXX01YYY",XXX02YYY") extract the numbers.

Usage

```
extract_differences_between_characters(strings)
```

Arguments

strings	vector of character strings for elements to be extracted from.

Value

vector of strings similar to the input vector but without shared leading and trailing characters.

Author(s)

Eike Luedeling

Examples

```
 extract_differences_between_characters(c("Temp_01","Temp_02","Temp_03"))
extract_differences_between_characters(c("Temp_01_Tmin","Temp_02_Tmin","Temp_03_Tmin"))
 extract_differences_between_characters(c("a","b"))
```

extract_temperatures_from_grids

Extract temperature information from gridded dataset

Description

Temperature data is often available in gridded format, and records for particular points must be extracted for work on site-specific issues (such as chill calculation). This function implements this, for certain types of gridded data.

Usage

```
extract_temperatures_from_grids(
  coordinates,
  grid_format,
  grid_specifications,
  scenario_year = NA,
  reference_year = NA,
  scenario_type = NA,
  labels = NA,
  temperature_check_args = NULL
)
```

Arguments

coordinates	numeric vector specifying coordinates for the point location of interest. These coordinates have to use the same coordinate system as the grids, from which data are to be extracted. The elements can be named as 'longitude' and 'latitude', or provided as unnamed elements. In the latter case, the first element is interpreted as the x-coordinate (e.g. longitude or Easting) and the second element as the y-coordinate (e.g. latitude or Northing).
grid_format	character string specifying the type of raster data. See details below.
grid_specifications	list of specifications that instruct the function on where to find the temperature grids. See grid_format descriptions for what is required here.
scenario_year	year the temperature scenario is representative of, e.g. 2050, 2080. If the scenario period is an interval, this should be the median of all years in this interval.
reference_year	year of reference for the gridded climate data. This is only important for relative temperature scenarios. If the reference period is an interval, this should be the median of all years in this interval.
scenario_type	character string specifying whether the climate data contains a relative or absolute temperature sceanario. Accordingly, this should be 'relative' or 'absolute'. Can also be NA, which is the default, in which case the function makes a guess on which type applies. This guess is directed by the temperature_check_args.
labels	list of labels to be passed to the labels argument of the resulting temperature_scenario
temperature_check_args	list of arguments to be passed to the check_temperature_scenario function. Check documentation of that function for details.

Details

The following climate data formats are supported: "AFRICLIM" - data downloaded from https://www.york.ac.uk/environmen "CCAFS" - data downloaded from http://ccafs-climate.org/data_spatial_downscaling/; "WorldClim" - data downloaded from http://www.worldclim.org/. All these databases provide separate zipped files for monthly minimum and monthly maximum temperatures, but they differ slightly in format and structure. If you want to see additional formats included, please send me a message.

Value

temperature scenario object extracted from the grids, consisting of the following elements: 'data' = a data frame with n_intervals elements containing the absolute or relative temperature information. 'reference_year' = the year the scenario is representative of. 'scenario_type' = the scenario type ('absolute' or 'relative'); 'labels' = and elements attached to the input temperature_scenario as an element names 'labels'.

The function generates errors, when problems arise.

Author(s)

Eike Luedeling

Examples

```
  coordinates<-c(10.6082,34.9411)
 # grid_specifications<-list(base_folder="D:/DATA/AFRICLIM/GeoTIFF_30s/future_scenarios/",
 #                    minfile="tasmin_rcp45_2055_CCCma-CanESM2_CCCma-CanRCM4_wc30s.zip",
 #                    maxfile="tasmax_rcp45_2055_CCCma-CanESM2_CCCma-CanRCM4_wc30s.zip")

 # extract_temperatures_from_grids(coordinates,grid_format="AFRICLIM",grid_specifications,
 #    scenario_type="relative",scenario_year=2055)

 # grid_specifications<-list(base_folder="D:/DATA/CCAFS_climate/",
 #                         minfile="bcc_csm1_1_rcp2_6_2030s_tmin_30s_r1i1p1_b4_asc.zip",
 #                         maxfile="bcc_csm1_1_rcp2_6_2030s_tmax_30s_r1i1p1_b4_asc.zip")
 #temps<-extract_temperatures_from_grids(coordinates,grid_format="CCAFS",grid_specifications,
 #                                       scenario_type="relative",scenario_year=2035)
```

`filter_temperatures` *Quality filter for temperature records*

Description

This function attempts to remove erroneous temperature readings. This is tricky because of the wide range of errors that can occur, so this isn't necessarily sufficient for problems of particular records.

Usage

```
filter_temperatures(
  temp_file,
  remove_value = NA,
  running_mean_filter = NA,
  running_mean_length = 3,
  min_extreme = NA,
  max_extreme = NA,
  max_missing_in_window = 1,
  missing_window_size = 9
)
```

Arguments

temp_file	file containing temperature data. Should have columns c("Year","Month","Day","Temp" - and "Hour" for hourly data).
remove_value	numeric value indicating 'no data'.
running_mean_filter	deviation from a running mean over all temperature data that identifies a value as an erroneous outlier.
running_mean_length	number of records to be included in a running mean.
min_extreme	lowest plausible temperature on the record. All lower ones are removed.
max_extreme	highest plausible temperature on the record. All higher ones are removed.
max_missing_in_window	maximum share of values (0..1) in a running window of size missing_window_size around each value that can be missing. If this is exceeded, the value is removed.
missing_window_size	size of the window used for checking for missing values.

Value

filtered temperature dataset, from which records identified as erroneous were removed.

Author(s)

Eike Luedeling

Examples

```
weather<-fix_weather(KA_weather[which(KA_weather$Year>2009),])

hourtemps<-stack_hourly_temps(weather, latitude=50.4)

filtered<-filter_temperatures(hourtemps$hourtemps,remove_value=-99,
  running_mean_filter=3)
```

---

fix_weather *Weather data fixer and quality checker*

Description

This function identifies and interpolates gaps in daily weather records

Usage

```
fix_weather(
  weather,
  start_year = 0,
  end_year = 3000,
  start_date = 1,
  end_date = 366,
  columns = c("Tmin", "Tmax"),
  end_at_present = TRUE
)
```

Arguments

weather	a data.frame containing a daily time series dataset. It should have columns c("Year", "Month", "Day") or c("YEAR", "MONTH","DAY") or "YEARMODA".
start_year	integer marking the first year of interest. If not specified, this is assumed to be year 0, which probably means that the entire record will be considered.
end_year	integer marking the last year of interest. If not specified, this is assumed to be year 3000, which probably means that the entire record will be considered.
start_date	start date of the sub-annual period of interest (e.g. the assumed chilling period), defaults to 1 (1st Jan) if not specified
end_date	end date of the sub-annual period of interest (e.g. the assumed chilling period), defaults to 366 (31st Dec, also in non-leap years) if not specified
columns	character vector containing the names of columns of the weather file that should be interpolated and quality checked. If not specified, this defaults to "Tmin" and "Tmax". If these columns don't exist, the function generates an error.
end_at_present	boolean variable indicating whether the interval of interest should end on the present day, rather than extending until the end of the year specified under time_interval[2] (if time_interval[2] is the current year).

Details

This function produces a complete record containing all dates between the 1st day of the start year and the last day of the end year (unless the first/last day of the record is after/before these dates - in that case the record is not extended). The values for the columns specified by the columns attribute are linearly interpolated. Missing values during the period indicated by start_date and end_date are added up and summarized in a quality control table.

Value

list with two elements: weather: contains the interpolated weather record QC: contains the quality control data.frame, which summarizes missing days, incomplete days (days on which any value is missing), and percentage completeness.

Author(s)

Eike Luedeling

Examples

```
fix_weather(KA_weather,2000,2010)

#use a subset of the KA_weather dataset and add an additional day after a gap
KA_weather_gap<-rbind(KA_weather,c(Year=2011,Month=3,Day=3,Tmax=26,Tmin=14))
#fill in the gaps
fix_weather(KA_weather_gap, 1990,2011,300,100)

#fix_weather(KA_weather)
```

GDD *Calculation of cumulative heat according to the Growing Degree Day Model*

Description

This function calculates heat for temperate trees according to the Growing Degree Day Model. Note that the calculuation differs slightly from the original, in which it is based on daily temperature extremes only. This equation here works with hourly temperatures. The normal GDD equation is GDD=(Tmax-Tmin)/2-Tbase, with Tmax=30 for Tmax>30, and Tmin=10 for Tmin<10. Tbase is a species-specific base temperature. The first part of the equation is the arithmetic mean of daily temperature extremes. In the present equation, this is replaced by Thourly/24 for each hourly temperature value. If chillR was using a triangular daily temperature curve, the result would be the same for both equations. Since chillR uses a sine function for daytime warming and a logarithmic decay function for nighttime cooling, however, there will be a slight deviation. This could be handled by defining a function the runs with daily weather data. chillR doesn't currently have this capability, since its primary focus is on metrics that require hourly data.

Usage

```
GDD(HourTemp, summ = TRUE, Tbase = 5)
```

Arguments

HourTemp	Vector of hourly temperatures.
summ	Boolean parameter indicating whether calculated metrics should be provided as cumulative values over the entire record (TRUE) or as the actual accumulation for each hour (FALSE).
Tbase	Base temperature, above which Growing Degrees accrue.

Details

Growing Degree Hours are calculated as suggested by Anderson et al. (1986).

Value

Vector of length length(HourTemp) containing the cumulative Growing Degree Days over the entire duration of HourTemp.

Author(s)

Eike Luedeling

References

Growing Degree Days reference:

http://agron-www.agron.iastate.edu/Courses/agron212/Calculations/GDD.htm

Examples

```
weather<-fix_weather(KA_weather[which(KA_weather$Year>2006),])

hourtemps<-stack_hourly_temps(weather,latitude=50.4)

GDD(hourtemps$hourtemps$Temp)
```

GDH *Calculation of cumulative heat according to the Growing Degree Hours Model*

Description

This function calculates heat for temperate trees according to the Growing Degree Hours Model.

Usage

```
GDH(HourTemp, summ = TRUE)
```

Arguments

HourTemp	Vector of hourly temperatures.
summ	Boolean parameter indicating whether calculated metrics should be provided as cumulative values over the entire record (TRUE) or as the actual accumulation for each hour (FALSE).

Details

Growing Degree Hours are calculated as suggested by Anderson et al. (1986).

Value

Vector of length length(HourTemp) containing the cumulative Growing Degree Hours over the entire duration of HourTemp.

Author(s)

Eike Luedeling

References

Growing Degree Hours reference:

Anderson JL, Richardson EA, Kesner CD (1986) Validation of chill unit and flower bud phenology models for 'Montmorency' sour cherry. Acta Hortic 184, 71-78

Examples

```
weather<-fix_weather(KA_weather[which(KA_weather$Year>2006),])

hourtemps<-stack_hourly_temps(weather,latitude=50.4)

GDH(hourtemps$hourtemps$Temp)
```

GDH_model — *Calculation of cumulative heat according to the Growing Degree Hours Model (alternative function name)*

Description

This function calculates heat for temperate trees according to the Growing Degree Hours Model.

Usage

```
GDH_model(HourTemp, summ = TRUE)
```

Arguments

HourTemp	Vector of hourly temperatures.
summ	Boolean parameter indicating whether calculated metrics should be provided as cumulative values over the entire record (TRUE) or as the actual accumulation for each hour (FALSE).

Details

Growing Degree Hours are calculated as suggested by Anderson et al. (1986).

Value

Vector of length length(HourTemp) containing the cumulative Growing Degree Hours over the entire duration of HourTemp.

Author(s)

Eike Luedeling

References

Growing Degree Hours reference:

Anderson JL, Richardson EA, Kesner CD (1986) Validation of chill unit and flower bud phenology models for 'Montmorency' sour cherry. Acta Hortic 184, 71-78

Examples

```
weather<-fix_weather(KA_weather[which(KA_weather$Year>2006),])

hourtemps<-stack_hourly_temps(weather,latitude=50.4)

GDH_model(hourtemps$hourtemps$Temp)
```

---

genSeason *Generate Seasons*

Description

Identify the hours, days or months in a (monthly, daily or hourly) temperature dataset that belong to a particular season. Seasons are defined according to the 'mrange' argument, which specifies the start and end month of the season. The 'years' argument specifies the year, in which the dormancy season of interest ends.

Usage

```
genSeason(temps, mrange = c(8, 6), years)
```

Arguments

temps	list. generated by 'chillR'
mrange	numeric. vector with two entries for the range of months (start month and end month)
years	numeric. vector of years to be considered (with each entry specifying the year, in which the season **ends**)

genSeasonList *genSeasonList*

Description

Generates a list with data.frame elements for each season.

Usage

```
genSeasonList(temps, mrange = c(8, 6), years)
```

Arguments

temps	data.frame. Must have columns 'Temp' containing the temperatures, 'JDay' the JDays, 'Month' the months and 'Year' the years. This kind of data frame is for instance generated by stack_hourly_temps, but can also be generated by hand or using a different routine.
mrange	numeric. vector of length two for the range of months the season should span. E.g. 'mrange=c(8,6)' would span a season from August to next June. There must not be any overlap in months, i.e. mrange[1] must be larger mrange[2].
years	numeric. vector of years to be considered

Value

Returns a list of data frames. Each element of the list corresponds to one season. The 'data.frame' for each year has named columns 'Temp', 'JDay' and 'Year'.

---

gen_rel_change_scenario *Generates relative climate change scenarios based on extracted CMIP6 data*

Description

Takes the extracted CMIP6 data and returns climate change scenarios, which can then be used to generate weather data.

Usage

```
gen_rel_change_scenario(
  downloaded_list,
  scenarios = c(2050, 2085),
  reference_period = c(1986:2014),
  future_window_width = 30
)
```

Arguments

`downloaded_list`
: list of data.frames, generated using the extract_cmip6_data function. Elements are named after the shared socioeconomic pathway ('SSP') and global climate model ('GCM')

`scenarios`
: numeric vector, states the future years, for which the climate change scenarios should be generated. By default set to c(2050, 2085).

`reference_period`
: numeric vector specifying the years to be used as the reference period. Defaults to c(1986:2014).

`future_window_width`
: numeric, sets the window width of the running mean calculation for the mean temperatures of the years indicated by scenarios

Value

data.frame for the calculated relative change scenarios, all locations, SSPs, timepoints, GCMs combined

Author(s)

Lars Caspersen

Examples

```
## Not run:
download_cmip6_ecmwfr(scenario = 'ssp1_2_6',
                      area = c(55, 5.5, 47, 15.1),
                      user = 'write user id here',
                      key = 'write key here',
                      model = 'AWI-CM-1-1-MR',
                      frequency = 'monthly',
                      variable = c('Tmin', 'Tmax'),
                      year_start = 2015,
                      year_end = 2100)

download_baseline_cmip6_ecmwfr(
    area = c(55, 5.5, 47, 15.1),
    user = 'write user id here',
    key = 'write key here',
    model = 'AWI-CM-1-1-MR',
    frequency = 'monthly',

station <- data.frame(
      station_name = c('Zaragoza', 'Klein-Altendorf', 'Sfax',
      'Cieza', 'Meknes', 'Santomera'),
      longitude = c(-0.88,  6.99, 10.75, -1.41, -5.54, -1.05),
      latitude = c(41.65, 50.61, 34.75, 38.24, 33.88, 38.06))

extracted <- extract_cmip6_data(stations = station)
```

```
gen_rel_change_scenario(extracted)


## End(Not run)
```

`getClimateWizardData` *Extract climate data from the ClimateWizard database*

Description

This function makes use of an API provided by the International Center for Tropical Agriculture (CIAT) to access climate scenario data for a location of interest. Climate model runs are queried and data returned and summarized according to the specified parameters. A number of metrics are available for several climate models, which are listed in the API repository. Refer to this document for details on what can be downloaded. This function provides the additional option of automatically retrieving all data referring to changes in daily temperature extremes (by month), by setting the "`metric`" parameter to "monthly_min_max_temps". It also offers the option to automatically obtain data for all climate models included in the database (as of January 2018).

Usage

```
getClimateWizardData(
  coordinates,
  scenario,
  start_year,
  end_year,
  baseline = c(1950, 2005),
  metric = "monthly_min_max_temps",
  GCMs = "all",
  temperature_generation_scenarios = FALSE
)
```

Arguments

`coordinates`	position of the point of interest, specified by a vector with two elements that are called longitude and latitude (e.g. `c(longitude = 10,latitude = 20)`).
`scenario`	representative concentration pathway scenario. Can only be "historical", "rcp45" or "rcp85".
`start_year`	start year of the interval, for which data is to be summarized.
`end_year`	end year of the interval, for which data is to be summarized.
`baseline`	numeric vector of length 2 indicating the time interval to be used as baseline for the climate scenario. The function then returns projected values relative to this baseline. Defaults to `c(1950, 2005)` for the standard baseline of the ClimateWizard dataset. This can also assume different values, but it must span an interval of at least 20 years within the [1950; 2005] interval. Needs to be set to NA for the function to return absolute values.

`metric` vector of metrics to output, from a list specified in the reference provided above. This can also be "monthly_min_max_temps", which returns all mean monthly minimum and maximum temperatures, or "precipitation" for precipitation data for all months, or "monthly_tmean" for the mean monthly temperatures of all months.

`GCMs` vector of GCMs to be accessed, from a list specified in the above reference. This can also be "all" for all available GCMs (as of January 2018).

`temperature_generation_scenarios`
parameter to indicate whether the scenarios to be generated should be formatted in such a way that they are directly usable by chillR's temperature_generation function. This is only applicable, when `metric == 'monthly_min_max_temps'`.

Value

data.frame containing the requested information.

Author(s)

Eike Luedeling

References

Girvetz E, Ramirez-Villegas J, Navarro C, Rodriguez C, Tarapues J, undated. ClimateWizard REST API for querying climate change data. https://github.com/CIAT-DAPA/climate_wizard_api

Examples

```
# the example is #d out, since the download request sometimes times out, and that
# causes problems with CRAN approval of the package

# getClimateWizardData(coordinates=c(longitude=10.613975,latitude=34.933439),
#   scenario="rcp45", start_year=2020, end_year=2050,
#   metric=c("CD18","R02"), GCMs=c("bcc-csm1-1","BNU-ESM"))
```

---

getClimateWizard_scenarios

Extract mutltiple scenarios from the ClimateWizard database

Description

This function is a wrapper for the getClimateWizardData function to access climate scenario data for a location of interest. Climate model runs are queried and data returned and summarized according to the specified parameters. A number of metrics are available for several climate models, which are listed in https://github.com/CIAT-DAPA/climate_wizard_api. This function can download data for multiple climate scenarios, saving users the effort to retrieve them separately.

Usage

```
getClimateWizard_scenarios(
  coordinates,
  scenarios,
  start_years,
  end_years,
  baseline = c(1950, 2005),
  metric = "monthly_min_max_temps",
  GCMs = "all"
)
```

Arguments

coordinates	position of the point of interest, specified by a vector with two elements that are called longitude and latitude (e.g. c(longitude=10, latitude=20)).
scenarios	vector of representative concentration pathway scenarios. Can only be "historical", "rcp45" or "rcp85".
start_years	vector of start year of the intervals, for which data is to be summarized. Must be of same length as scenarios.
end_years	vector of end years of the intervals, for which data is to be summarized. Must be of same length as scenarios.
baseline	numeric vector of length 2 indicating the time interval to be used as baseline for the climate scenario. The function then returns projected values relative to this baseline. Defaults to c(1950,2005) for the standard baseline of the ClimateWizard dataset. This can also assume different values, but it must span an interval of at least 20 years within the [1950; 2005] interval. Needs to be set to NA for the function to return absolute values.
metric	vector of metrics to output, from a list specified in the reference provided above. This can also be "monthly_min_max_temps", which returns all mean monthly minimum and maximum temperatures, or "precipitation" for precipitation data for all months, or "monthly_tmean" for the mean monthly temperatures of all months.
GCMs	vector of GCMs to be accessed, from a list specified in the above reference. This can also be "all" for all available GCMs (as of January 2018).

Details

Note that this function lacks quality checks. If something goes wrong, you may consider checking individual scenarios with the getClimateWizardData function.

Value

data.frame containing the requested information.

Author(s)

Eike Luedeling

References

Girvetz E, Ramirez-Villegas J, Navarro C, Rodriguez C, Tarapues J, undated. ClimateWizard REST API for querying climate change data. https://github.com/CIAT-DAPA/climate_wizard_api

Examples

```
#example is #d out, because of runtime issues.
#getC<-getClimateWizard_scenarios(coordinates=c(longitude=6.99,latitude=50.62),
#                                 scenarios=c("rcp85","rcp45"),
#                                 start_years=c(2070,2035),
#                                 end_years=c(2100,2065),
#                                 metric=c("monthly_tmean"),
#                                 GCMs=c("all"))
```

---

get_last_date *Get the last date from a phenology record*

Description

When looking at multi-year phenology records, it is normally obvious in which year bloom occurred last. Determining this with an automated procedure, however, is a bit tricky, when the range of phenological dates spans across a calendar year transition. This function finds the latest phenological date of the record. This is the date before the longest phenological date gap.

Usage

```
get_last_date(dates, first = FALSE)
```

Arguments

dates	numeric vector of Julian dates (days of the year)
first	boolean variable that can be set to TRUE to get the first, not the last, date of the phenology record.

Value

the latest (earliest) date of the series, under the assumption that the longest period without bloom can be interpreted as separating the phenological seasons. This should be a reasonable assumption in most cases.

Author(s)

Eike Luedeling

Examples

```
get_last_date(c(1,3,6,8,10,25))
get_last_date(c(345,356,360,365,2,5,7,10))
get_last_date(c(345,356,360,365,2,5,7,10),first=TRUE)
```

`get_weather` *Download weather data from online database*

Description

This function retrieves either a list of nearby weather stations for a specified point location, or it downloads weather data for a specific weather station.

Usage

```
get_weather(
  location,
  time_interval = NA,
  database = "UCIPM",
  station_list = NULL,
  stations_to_choose_from = 25,
  end_at_present = TRUE
)
```

Arguments

`location`	either a vector of geographic coordinates, or the 'chillRcode' of a weather station in the specified database. See details.
`time_interval`	numeric vector with two elements, specifying the start and end date of the period of interest.
`database`	the database to be accessed. Must be "GSOD", "CIMIS" or "UCIPM". Since among these, "UCIPM" is the most comprehensive one for California, the initial area of interest, this is the default.
`station_list`	if the list of weather stations has already been downloaded, the list can be passed to the function through this argument. This can save a bit of time, since it can take a bit of time to download the list, which can have several MB.
`stations_to_choose_from`	if the location is specified by geographic coordinates, this argument determines the number of nearby stations in the list that is returned.
`end_at_present`	boolean variable indicating whether the interval of interest should end on the present day, rather than extending until the end of the year specified under time_interval[2] (if time_interval[2] is the current year).

Details

weather databases, from which chillR can download data: NOAA NCDC Global Summary of the Day - "GSOD" (https://data.noaa.gov/dataset/global-surface-summary-of-the-day-gsod)

California Irrigation Management Information System (CIMIS) - "CIMIS" (http://www.cimis.water.ca.gov/)

University of California Integrated Pest Management (UCIPM) - "UCIPM" (http://ipm.ucdavis.edu/WEATHER/)

several formats are possible for specifying the location vector, which can consist of either two or three coordinates (it can include elevation). Possible formats include c(1,2,3), c(1,2), c(x=1,y=2,z=3), c(lat=2,long=1,elev=3). If elements of the vector are not names, they are interpreted as c(Longitude, Latitude, Elevation).

The 'chillRCode' is generated by this function, when it is run with geographic coordinates as location inputs. In the list of nearby stations that is returned then, the chillRCode is provided and can then be used as input for running the function in 'downloading' mode. For downloading the data, use the same call as before but replace the location argument with the chillRCode.

Value

The output depends on how the location is provided. If it is a coordinate vector, the function returns a list of station_to_choose_from weather stations that are close to the specified location. This list also contains information about how far away these stations are (in km), how much the elevation difference is (if elevation is specified; in m) and how much overlap there is between the data contained in the database and the time period specified by time_interval.

Note

Many databases have data quality flags, which may sometimes indicate that data aren't reliable. These are not considered by this function!

see the documentation of the handler functions (e.g. handle_ucipm) for details.

Author(s)

Eike Luedeling

References

The chillR package:

Luedeling E, Kunz A and Blanke M, 2013. Identification of chilling and heat requirements of cherry trees - a statistical approach. International Journal of Biometeorology 57,679-689.

Examples

```
#stat_list<-handle_gsod(action="list_stations",location=c(x=-122,y=38.5),
#  time_interval=c(2002,2002))
#the line above takes longer to run than CRAN allows for examples. The line below therefore
#generates an abbreviated stat_list that allows running the code.
stat_list<-data.frame(chillR_code=c("724828_99999","724828_93241","720576_174"),
   Lat=c(38.383,38.378,38.533),Long=c(-121.967,-121.958,-121.783),
   BEGIN=c(20010811,20060101,20130101),END=c(20051231,20160110,20160109))
```

```
#gw<-get_weather(location="724828_93241",time_interval=c(2012,2012),database="GSOD",
#   station_list = stat_list)

#stat_list<-get_weather(location=c(lat=50,lon=10,ele=150),time_interval=c(2001,2001),
#  database="UCIPM")
#chillRcode<-stat_list[which(stat_list$Perc_interval_covered==
#max(stat_list$Perc_interval_covered)),"chillR_code"][1]
  #after the first few lines here, the code should be "CEDARVIL.C"
#gw<-get_weather(location="CEDARVIL.C",time_interval=c(2001,2001),database="UCIPM")
#weather<-weather2chillR(gw,"GSOD")
#make_chill_plot(tempResponse(stack_hourly_temps(fix_weather(weather))),
#           "Chill_Portions",start_year=2005,end_year=2011,metriclabel="Chill Portions")
```

---

handle_cimis *List, download or convert to chillR format data from the CIMIS database*

Description

This function can do three things related to the California Irrigation Management Information System ("CIMIS") database: 1. it can list stations that are close to a specified position (geographic coordinates) 2. it can retrieve weather data for a named weather station 3. it can 'clean' downloaded data, so that they can easily be used in chillR Which of these functions is carried out depends on the action argument.

Usage

```
handle_cimis(
  action,
  location = NA,
  time_interval = NA,
  station_list = NULL,
  stations_to_choose_from = 25,
  drop_most = TRUE,
  end_at_present = TRUE
)
```

Arguments

action	if this is the character string "list_stations", the function will return a list of the weather stations from the database that are closest to the geographic coordinates specified by location. if this is the character string "download_weather", the function will attempt to download weather data from the database for the station named by the location argument, which should then be a character string corresponding to the chillRcode of the station (which you can get by running this function in 'list_stations mode) if this is a downloaded weather file (downloaded

by running this function in 'download weather' mode), the function cleans the file and makes it ready for use in chillR. If the input is just a dataframe (not a list, as produced with this function), you have to specify the database name with the database argument

`location` either a vector of geographic coordinates (for the 'list_stations' mode), or the 'chillRcode' of a weather station in the specified database (for the 'download_weather' mode. When running this function for data cleaning only, this is not needed.

`time_interval` numeric vector with two elements, specifying the start and end date of the period of interest. Only required when running in 'list_stations' or 'download weather' mode

`station_list` if the list of weather stations has already been downloaded, the list can be passed to the function through this argument. This can save a bit of time, since it can take a bit of time to download the list, which can have several MB.

`stations_to_choose_from`
if the location is specified by geographic coordinates, this argument determines the number of nearby stations in the list that is returned.

`drop_most` boolean variable indicating if most columns should be dropped from the file. If set to TRUE (default), only essential columns for running chillR functions are retained.

`end_at_present` boolean variable indicating whether the interval of interest should end on the present day, rather than extending until the end of the year specified under time_interval[2] (if time_interval[2] is the current year).

Details

This function can run independently, but it is also called by the get_weather and weather2chillR functions, which some users might find a bit easier to handle.

The CIMIS dataset is described here: http://www.cimis.water.ca.gov/

Under the 'list_stations' mode, several formats are possible for specifying the location vector, which can consist of either two or three coordinates (it can include elevation). Possible formats include c(1,2,3), c(1,2), c(x=1,y=2,z=3), c(lat=2,long=1,elev=3). If elements of the vector are not names, they are interpreted as c(Longitude, Latitude, Elevation).

The 'chillRCode' is generated by this function, when it is run with geographic coordinates as location inputs. In the list of nearby stations that is returned then, the chillRCode is provided and can then be used as input for running the function in 'downloading' mode. For downloading the data, use the same call as before but replace the location argument with the chillRCode.

Value

The output depends on the action argument. If it is 'list_stations', the function returns a list of station_to_choose_from weather stations that are close to the specified location. This list also contains information about how far away these stations are (in km), how much the elevation difference is (if elevation is specified; in m) and how much overlap there is between the data contained in the database and the time period specified by time_interval. If action is 'download_weather' the output is a list of two elements: 1. database="CIMIS" 2. the downloaded weather record, extended to the full duration of the specified time interval. If action is a weather data.frame or a weather

record downloaded with this function (in 'download_weather' mode), the output is the same data in a format that is easy to use in chillR. If drop_most was set to TRUE, most columns are dropped.

Note

Many databases have data quality flags, which may sometimes indicate that data aren't reliable. These are not considered by this function!

Past CIMIS data is provided to the public as compressed data files of annual data, which contain data for all stations for the respective years. The same strategy was followed for monthly data of the past year. This means that in order to get to the records for one given station, it is necessary to download data for all stations first, before extracting weather for the station of interest. This means that downloads take a lot longer than one might expect, and the downloaded data volume is a multiple of what is really of interest.

Author(s)

Eike Luedeling

References

The chillR package:

Luedeling E, Kunz A and Blanke M, 2013. Identification of chilling and heat requirements of cherry trees - a statistical approach. International Journal of Biometeorology 57,679-689.

Examples

```
# the example is #d out, since the download request sometimes times out, and that
# causes problems with CRAN approval of the package

# handle_cimis(action = "list_stations",
#              location = c(x = -122, y = 38.5),
#              time_interval = c(2012, 2012))

# stat_list <- data.frame("Station Number" = c("119", "139", "6"),
#                         Latitude = c(38.49500, 38.50126, 38.53569),
#                         Longitude = c(-122.0040, -121.9785, -121.7764),
#                    Start_date =c("1993-08-21 UTC", "1998-06-15 UTC", "1982-07-17 UTC"),
#                         End_date = c("1995-01-25", "2016-03-06", "2016-03-06"))

# gw <- handle_cimis(action = "download_weather",
#                    location = "6",
#                    time_interval = c(1982, 1982),
#                    station_list = stat_list)

# weather <- handle_cimis(gw)

# make_chill_plot(tempResponse(stack_hourly_temps(fix_weather(weather)),
#                 Start_JDay = 300, End_JDay = 50),
#                 "Chill_Portions", start_year = 2010, end_year = 2012,
#                 metriclabel = "Chill Portions", misstolerance = 50)
```

handle_dwd *List, download or convert to chillR format data from the Deutscher Wetterdienst database*

Description

This function accesses the Deutscher Wetterdienst database and allows to:

- 1) list a number of weather stations that are close to a specific position (geographic coordinates)
- 2) obtain weather data for one or more weather stations through the station ID
- 3) 'clean' and 'format' downloaded data, so the records can easily be used in other chillR functions

Usage

```
handle_dwd(
  action,
  location = NA,
  time_interval = c(19160101, Date2YEARMODA(Sys.Date())),
  station_list = NULL,
  stations_to_choose_from = 25,
  drop_most = TRUE,
  end_at_present = TRUE,
  add.DATE = FALSE,
  quiet = FALSE,
  add_station_name = FALSE
)
```

Arguments

`action` is a character string to decide on 3 modes of action for the function.

- *'list_stations'* returns a data frame with the information on weather stations that are to the location defined by `number_of_stations` and `location` parameters.
- *'download_weather'* retrieves the records for one or more weather stations defined in the `location` parameter.
- If the input is a data frame previously downloaded with the mode *'download_weather'*, the function will format the data frame using the chillR structure.

`location` accepts a numeric vector with two or three elements representing the longitude, latitude, and elevation of a given place or a vector of character strings representing the ID of the weather stations of interest. If `action = 'list_stations'`, `location` requires the coordinates of the place and optionally the elevation. This vector can be named or not. Valid names are: `'y'`, `'Y'`, `'latitude'`, `'lat'`, `'Latitude'`, `'Lat'`, `'LATITUDE'`, `'LAT'` for latitude, `'x'`, `'X'`, `'longitude'`,

'long', 'Longitude', 'Long', 'LONGITUDE', 'LONG' for longitude, and 'z', 'Z', 'elevation', 'elev', 'Elevation', 'Elev', 'ELEVATION', 'ELEV' for elevation. If action = 'download_weather', location accepts the ID of the station as character string.

time_interval numeric vector with two elements, specifying the start and end date of the period of interest. Only required when running in 'list_stations' or 'download_weather' mode. Unlike other functions from the handle family, handle_dwd allows specifying the date in YEARMODA format. Default is set to 19160101 (the earliest date on record) and the current date.

station_list accepts a data frame if the list of weather stations has already been downloaded. The list can be passed to the function through this argument. This can save a bit of time, since it can take a bit of time to download the list, which can have several MB.

stations_to_choose_from
if the location is specified by geographic coordinates, this argument determines the number of nearby stations in the list that is returned.

drop_most boolean variable indicating if most columns should be dropped from the file if a list of data frames is provided to the action argument. If set to TRUE (the default), only essential columns for running chillR functions are retained.

end_at_present boolean variable indicating whether the interval of interest should end on the present day, rather than extending until the end of the year specified under time_interval[2] (if time_interval[2] is the current year). **DEPRECATED** in this function since time_interval already allows specifying the present day.

add.DATE is a boolean parameter to be passed to make_all_day_table if action is a collection of outputs (in the form of list) from the function in the downloading format.

quiet is a boolean parameter to be passed to download.file if action = "download_weather".

add_station_name
is a boolean parameter to include the name of the respective weather station in the resulting data frame in case the function is used in the downloading or formatting mode.

Value

If action = 'list_stations', the function returns a data frame with 'stations_to_choose_from' rows and 9 columns. This data frame contains information about the weather stations (Latitude, Longitude, among others). If action = 'download_weather', the function returns a list of length according to the length of the location parameter. Each list elements is a data frame containing the data downloaded from the database. If the action is provided with the list generated by the function in the downloading mode, the function will return a list of data frames structured according to the chillR format. If drop_most is set to TRUE, the function will keep only the most relevant variables for standard chillR analyses.

Note

Many databases have data quality flags, which may sometimes indicate that data aren't reliable. These are not considered by this function!

Author(s)

Eduardo Fernandez and Eike Luedeling

References

Fernandez, E., Whitney, C., and Luedeling, E. 2020. The importance of chill model selection - A multi-site analysis. European Journal Of Agronomy 119: 126103

Examples

```
# The following lines may take longer than required to pass the
# CRAN checks. Please, un-comment them to run the example

# stations <- handle_dwd(action = "list_stations",
#                        location = c(latitude = 53.5373, longitude = 9.6397),
#                        time_interval = c(20000101, 20101231),
#                        stations_to_choose_from = 25)

# data <- handle_dwd(action = "download_weather",
#                    location = stations[1 : 3, "Station_ID"],
#                    time_interval = c(20000101, 20020601),
#                    stations_to_choose_from = 25,
#                    station_list = stations,
#                    drop_most = TRUE,
#                    add.DATE = FALSE,
#                    quiet = TRUE,
#                    add_station_name = FALSE)

# data_modified <- handle_dwd(data, add.DATE = TRUE, drop_most = TRUE)
```

---

handle_dwd_old *List, download or convert to chillR format data from the Deutscher Wetterdienst database*

Description

This function is deprecated and will disappear soon. Please refer to the handle_dwd function for the most current functionality.

Usage

```
handle_dwd_old(
  action,
  location = NA,
  time_interval = c(19160101, Date2YEARMODA(Sys.Date())),
  station_list = NULL,
  stations_to_choose_from = 25,
  drop_most = TRUE,
```

```
  end_at_present = TRUE,
  add.DATE = FALSE,
  quiet = FALSE,
  add_station_name = FALSE
)
```

Arguments

action	is a character string to decide on 3 modes of action for the function. • *'list_stations'* returns a data frame with the information on weather stations that are to the location defined by number_of_stations and location parameters. • *'download_weather'* retrieves the records for one or more weather stations defined in the location parameter. • If the input is a data frame previously downloaded with the mode *'download_weather'*, the function will format the data frame using the chillR structure.
location	accepts a numeric vector with two or three elements representing the longitude, latitude, and elevation of a given place or a vector of character strings representing the ID of the weather stations of interest. If action = 'list_stations', location requires the coordinates of the place and optionally the elevation. This vector can be named or not. Valid names are: 'y', 'Y', 'latitude', 'lat', 'Latitude', 'Lat', 'LATITUDE', 'LAT' for latitude, 'x', 'X', 'longitude', 'long', 'Longitude', 'Long', 'LONGITUDE', 'LONG' for longitude, and 'z', 'Z', 'elevation', 'elev', 'Elevation', 'Elev', 'ELEVATION', 'ELEV' for elevation. If action = 'download_weather', location accepts the ID of the station as character string.
time_interval	numeric vector with two elements, specifying the start and end date of the period of interest. Only required when running in 'list_stations' or 'download_weather' mode. Unlike other functions from the handle family, handle_dwd allows specifying the date in YEARMODA format. Default is set to 19160101 (the earliest date on record) and the current date.
station_list	accepts a data frame if the list of weather stations has already been downloaded. The list can be passed to the function through this argument. This can save a bit of time, since it can take a bit of time to download the list, which can have several MB.
stations_to_choose_from	if the location is specified by geographic coordinates, this argument determines the number of nearby stations in the list that is returned.
drop_most	boolean variable indicating if most columns should be dropped from the file if a list of data frames is provided to the action argument. If set to TRUE (the default), only essential columns for running chillR functions are retained.
end_at_present	boolean variable indicating whether the interval of interest should end on the present day, rather than extending until the end of the year specified under time_interval[2] (if time_interval[2] is the current year). **DEPRECATED** in this function since time_interval already allows specifying the present day.

add.DATE is a boolean parameter to be passed to make_all_day_table if action is a collection of outputs (in the form of list) from the function in the downloading format.

quiet is a boolean parameter to be passed to download.file if action = "download_weather".

add_station_name is a boolean parameter to include the name of the respective weather station in the resulting data frame in case the function is used in the downloading or formatting mode.

Details

This function accesses the Deutscher Wetterdienst database and allows to:

- 1) list a number of weather stations that are close to a specific position (geographic coordinates)
- 2) obtain weather data for one or more weather stations through the station ID
- 3) 'clean' and 'format' downloaded data, so the records can easily be used in other chillR functions

Value

If action = 'list_stations', the function returns a data frame with 'stations_to_choose_from' rows and 9 columns. This data frame contains information about the weather stations (Latitude, Longitude, among others). If action = 'download_weather', the function returns a list of length according to the length of the location parameter. Each list, is a list of two elements; a data frame containing the data downloaded from the database and character string representing the respective database ('dwd'). If the action is provided with the list generated by the function in the downloading mode, the function will return a list of data frames structured according to the chillR format. If drop_most is set to TRUE, the function will keep only the relevant variables.

Note

Many databases have data quality flags, which may sometimes indicate that data aren't reliable. These are not considered by this function!

Author(s)

Eduardo Fernandez and Eike Luedeling

References

Fernandez, E., Whitney, C., and Luedeling, E. 2020. The importance of chill model selection - A multi-site analysis. European Journal Of Agronomy 119: 126103

Examples

```
# The following lines may take longer than required to pass the
# CRAN checks. Please, un-comment them to run the example
```

```
# stations <- handle_dwd_old(action = "list_stations",
#                         location = c(latitude = 53.5373, longitude = 9.6397),
#                         time_interval = c(20000101, 20101231),
#                         stations_to_choose_from = 25)

# data <- handle_dwd_old(action = "download_weather",
#                       location = stations[1 : 3, "Station_ID"],
#                       time_interval = c(20000101, 20020601),
#                       stations_to_choose_from = 25,
#                       station_list = stations,
#                       drop_most = TRUE,
#                       add.DATE = FALSE,
#                       quiet = TRUE,
#                       add_station_name = FALSE)

# data_modified <- handle_dwd_old(data, add.DATE = TRUE, drop_most = TRUE)
```

---

handle_gsod *List, download or convert to chillR format data from the Global Summary of the Day database*

Description

This function can do four things related to the Global Summary of the Day ("GSOD") database from the National Climatic Data Centre (NCDC) of the National Oceanic and Atmospheric Administration (NOAA):

- 1. It can list stations that are close to a specified position (geographic coordinates).
- 2. It can retrieve weather data for a named weather station (or a vector of multiple stations). For the name, the chillRcode from the list returned by the `list_stations` operation should be used.
- 3. It can 'clean' downloaded data (for one or multiple stations), so that they can easily be used in chillR
- 4. It can delete the downloaded intermediate weather files from the machine

  Which of these functions is carried out depends on the `action` argument.

This function can run independently, but it is also called by the `get_weather` and `weather2chillR` functions, which some users might find a bit easier to handle.

Usage

```
handle_gsod(
  action,
  location = NULL,
  time_interval = c(1950, 2020),
  stations_to_choose_from = 25,
  end_at_present = FALSE,
```

```
  add.DATE = FALSE,
  update_station_list = FALSE,
  path = "climate_data",
  update_all = FALSE,
  clean_up = NULL,
  override_confirm_delete = FALSE,
  max_distance = 150,
  min_overlap = 0,
  verbose = "normal"
)
```

Arguments

action	accepts 4 types of inputs to decide on the mode of action for the function. • if this is the character string "list_stations", the function will return a list of the weather stations from the database that are closest to the geographic coordinates specified by location. • if this is the character string "download_weather", the function will attempt to download weather data from the database for the station named by the location argument, which should then be a character string corresponding to the chillRcode of the station (which you can get by running this function in 'list_stations' mode). • if this is the character string "delete", the function will attempt to remove the intermediate downloaded weather data, which was saved in the folder specified by "path" argument. • if this is a collection of outputs obtained by running this function in the 'download weather' mode), the function cleans the weather files and make them ready for use in chillR. If the input is just a dataframe (not a list, as produced with this function), you have to specify the database name with the database argument.
location	either a vector of geographic coordinates (for the 'list_stations' mode), or the 'chillRcode' of a weather station in the specified database (for the 'download_weather' mode). When running this function for data cleaning only, this is not needed. For the 'download_weather' mode, this can also be a vector of 'chillRcodes', in which case records for all stations will be downloaded. The data cleaning mode can also handle a list of downloaded weather datasets.
time_interval	numeric vector with two elements, specifying the start and end date of the period of interest. Only required when running in 'list_stations' or 'download_weather' mode. The default is c(1950,2020).
stations_to_choose_from	if the location is specified by geographic coordinates, this argument determines the number of nearby stations in the list that is returned.
end_at_present	boolean variable indicating whether the interval of interest should end on the present day, rather than extending until the end of the year specified under time_interval[2] (if time_interval[2] is the current year).
add.DATE	is a boolean parameter to be passed to make_all_day_table if action is a collection of outputs (in the form of list) from the function in the downloading format.

update_station_list
: boolean, by default set FALSE. Decides if the weather station list is read from the disk (if present) or if it is newly downloaded in case of action = list_stations.

path
: character, by default "climate_data". Specifies the folder, relative to the working directory where the weather data is downloaded to.

update_all
: boolean, by default set to FALSE. If set TRUE, it will download every stations data, even if previously downloaded and still present in the temporary folder, specifief by the function argument path. If set FALSE, already downloaded years of a station will be skipped when download action is carried out again.

clean_up
: character, by default set to NULL. In combination with 'action = delete', this can be set to 'all' to delete all weather data, or 'station' if only data from specific stations ('location') should be deleted

override_confirm_delete
: Boolean, request whether the delete function needs user confirmation to run. Defaults to FALSE, and Should be set to TRUE if the function needs to be run without user intervention.

max_distance
: numeric, by default 150. Expresses the distance in kilometers how far away weather stations can be located from the original location, when searching for weather stations

min_overlap
: numeric, by default set to 0. Expresses in percent how much of the specified period needs to be covered by weather station to be included in the list, when searching for stations.

verbose
: is a character, deciding how much information is returned while downloading the weather data. By default set to "normal". If set to "detailed" the function will say how many years of data have been successfully downloaded for each station. If set "quiet" no information is printed during download.

Details

The GSOD database is described here: https://www.ncei.noaa.gov/access/metadata/landing-page/bin/iso?id=gov.noaa.ncdc:C00516

under the 'list_stations' mode, several formats are possible for specifying the location vector, which can consist of either two or three coordinates (it can include elevation). Possible formats include c(1, 2, 3), c(1, 2), c(x = 1, y = 2, z = 3), c(lat = 2, long = 1, elev = 3). If elements of the vector are not names, they are interpreted as c(Longitude, Latitude, Elevation).

The 'chillRCode' is generated by this function, when it is run with geographic coordinates as location inputs. In the list of nearby stations that is returned then, the chillRCode is provided and can then be used as input for running the function in 'downloading' mode. For downloading the data, use the same call as before but replace the location argument with the chillRCode.

Value

The output depends on the action argument. If it is 'list_stations', the function returns a list of station_to_choose_from weather stations that are close to the specified location. This list also contains information about how far away these stations are (in km), how much the elevation difference is (if elevation is specified; in m) and how much overlap there is between the data contained in

the database and the time period specified by time_interval. If action is 'download_weather' the output is a list of the downloaded weather record, extended to the full duration of the specified time interval. If the location input was a vector of stations, the output will be a list of such objects. If action is a weather data.frame or a weather record downloaded with this function (in 'download_weather' mode), the data structure remains in the same, but the data are processed for easy use with chillR. If drop_most was set to TRUE, most columns are dropped. If the location input was a list of weather datasets, all elements of the list will be processed. **IMPORTANT NOTE:** as of chillR version 0.73, the output format no longer contains a list element that specifies the database name, because this has been considered confusing (and annoying) by various users. This means, however, that some earlier calls to results from the handle_gsod function may produce errors now. Also note that a few parameters, station_list, drop_most, quiet, add_station_name are no longer needed due to some reworking of the function's mechanisms. After careful consideration, we decided to drop these parameters entirely, which may lead to some downward compatibility problems. Apologies for any inconvenience caused by this transition. If you want to keep using the previous function (which is much slower), feel free to adopt the deprecated handle_gsod_old function - but note that this will no longer be updated and may disappear eventually.

Note

Many databases have data quality flags, which may sometimes indicate that data aren't reliable. These are not considered by this function!

For many places, the GSOD database is quite patchy, and the length of the record indicated in the summary file isn't always very useful (e.g. there could only be two records for the first and last date). Files are downloaded by year, so if we specify a long interval, this may take a bit of time.

Author(s)

Adrian Fülle, Lars Caspersen, Eike Luedeling

References

The chillR package:

Luedeling E, Kunz A and Blanke M, 2013. Identification of chilling and heat requirements of cherry trees - a statistical approach. International Journal of Biometeorology 57,679-689.

Examples

```
#coordinates of Bonn
long <- 7.0871843
lat <- 50.7341602

#get a list of close-by weather stations
# stationlist <-
#   handle_gsod(action = "list_stations",
#               time_interval = c(1995,2000),
#               location = c(long,lat))

#download data
# test_data <-
```

```
#   handle_gsod(action = "download_weather",
#               time_interval = c(1995,2000),
#               location = stationlist$chillR_code[c(1,2)])
#
# format downloaded data
# test_data_clean <- handle_gsod(action = test_data)

## data deletion on disk for clean_up

# functions will ask for confirmation in the console - 'y' for yes to
# confirm deletion, anything else cancels the deletion

# handle_gsod(action = "delete",
#             clean_up = "all",
#             override_confirm_delete = TRUE)
```

`handle_gsod_old` *Deprecated version of handle_gsod. List, download or convert to chillR format data from the Global Summary of the Day database*

Description

This function is deprecated, but it will be retained for a few generations of updates. Its functionality has been fully replaced by the new version of the `handle_gsod` function, which does the same job, but much faster. That's probably the function you really want to use.

This function can do three things related to the Global Summary of the Day ("GSOD") database from the National Climatic Data Centre (NCDC) of the National Oceanic and Atmospheric Administration (NOAA):

- 1. It can list stations that are close to a specified position (geographic coordinates).
- 2. It can retrieve weather data for a named weather station (or a vector of multiple stations). For the name, the chillRcode from the list returned by the `list_stations` operation should be used.
- 3. It can 'clean' downloaded data (for one or multiple stations), so that they can easily be used in chillR

  Which of these functions is carried out depends on the action argument.

This function can run independently, but it is also called by the `get_weather` and `weather2chillR` functions, which some users might find a bit easier to handle.

Usage

```
handle_gsod_old(
  action,
  location = NA,
  time_interval = NA,
  station_list = NULL,
```

```
  stations_to_choose_from = 25,
  drop_most = TRUE,
  end_at_present = TRUE,
  add.DATE = TRUE,
  quiet = FALSE,
  add_station_name = FALSE
)
```

Arguments

action	accepts 3 types of inputs to decide on the mode of action for the function. • if this is the character string "list_stations", the function will return a list of the weather stations from the database that are closest to the geographic coordinates specified by location. • if this is the character string "download_weather", the function will attempt to download weather data from the database for the station named by the location argument, which should then be a character string corresponding to the chillRcode of the station (which you can get by running this function in 'list_stations' mode). • if this is a collection of outputs obtained by running this function in the 'download weather' mode), the function cleans the weather files and make them ready for use in chillR. If the input is just a dataframe (not a list, as produced with this function), you have to specify the database name with the database argument.
location	either a vector of geographic coordinates (for the 'list_stations' mode), or the 'chillRcode' of a weather station in the specified database (for the 'download_weather' mode). When running this function for data cleaning only, this is not needed. For the 'download_weather' mode, this can also be a vector of 'chillRcodes', in which case records for all stations will be downloaded. The data cleaning mode can also handle a list of downloaded weather datasets.
time_interval	numeric vector with two elements, specifying the start and end date of the period of interest. Only required when running in 'list_stations' or 'download_weather' mode.
station_list	if the list of weather stations has already been downloaded, the list can be passed to the function through this argument. This can save a bit of time, since it can take a bit of time to download the list, which can have several MB.
stations_to_choose_from	if the location is specified by geographic coordinates, this argument determines the number of nearby stations in the list that is returned.
drop_most	boolean variable indicating if most columns should be dropped from the file. If set to TRUE (default), only essential columns for running chillR functions are retained.
end_at_present	boolean variable indicating whether the interval of interest should end on the present day, rather than extending until the end of the year specified under time_interval[2] (if time_interval[2] is the current year).

add.DATE is a boolean parameter to be passed to make_all_day_table if action is a collection of outputs (in the form of list) from the function in the downloading format.

quiet is a boolean parameter to be passed to download.file if action = "download_weather".

add_station_name is a boolean parameter to include the name of the respective weather station in the resulting data frame in case the function is used in the downloading or formatting mode.

Details

The GSOD database is described here: https://www.ncei.noaa.gov/access/metadata/landing-page/bin/iso?id=gov.noaa.ncdc:C00516

under the 'list_stations' mode, several formats are possible for specifying the location vector, which can consist of either two or three coordinates (it can include elevation). Possible formats include c(1, 2, 3), c(1, 2), c(x = 1, y = 2, z = 3), c(lat = 2, long = 1, elev = 3). If elements of the vector are not names, they are interpreted as c(Longitude, Latitude, Elevation).

The 'chillRCode' is generated by this function, when it is run with geographic coordinates as location inputs. In the list of nearby stations that is returned then, the chillRCode is provided and can then be used as input for running the function in 'downloading' mode. For downloading the data, use the same call as before but replace the location argument with the chillRCode.

Value

The output depends on the action argument. If it is 'list_stations', the function returns a list of station_to_choose_from weather stations that are close to the specified location. This list also contains information about how far away these stations are (in km), how much the elevation difference is (if elevation is specified; in m) and how much overlap there is between the data contained in the database and the time period specified by time_interval. If action is 'download_weather' the output is a list of two elements: 1. database="GSOD" 2. the downloaded weather record, extended to the full duration of the specified time interval. If the location input was a vector of stations, the output will be a list of such objects. If action is a weather data.frame or a weather record downloaded with this function (in 'download_weather' mode), the output is the same data in a format that is easy to use in chillR. If drop_most was set to TRUE, most columns are dropped. If the location input was a list of weather datasets, all elements of the list will be processed.

Note

Many databases have data quality flags, which may sometimes indicate that data aren't reliable. These are not considered by this function!

For many places, the GSOD database is quite patchy, and the length of the record indicated in the summary file isn't always very useful (e.g. there could only be two records for the first and last date). Files are downloaded by year, so if we specify a long interval, this may take a bit of time.

Author(s)

Eike Luedeling and Eduardo Fernandez

References

The chillR package:

Luedeling E, Kunz A and Blanke M, 2013. Identification of chilling and heat requirements of cherry trees - a statistical approach. International Journal of Biometeorology 57,679-689.

Examples

```
# List the near weather stations
# stat_list <- handle_gsod_old(action = "list_stations",
#                           location = c(x = -122, y = 38.5),
#                           time_interval = c(2002, 2002))

# the line above takes longer to run than CRAN allows for examples.
# The line below therefore
# generates an abbreviated stat_list that allows running the code.

# stat_list <- data.frame(chillR_code = c("724828_99999",
#                                         "724828_93241",
#                                         "720576_174"),
#                         STATION.NAME = c("NUT TREE",
#                                          "NUT TREE AIRPORT",
#                                          "UNIVERSITY AIRPORT"),
#                         Lat = c(38.383, 38.378, 38.533),
#                         Long = c(-121.967, -121.958, -121.783),
#                         BEGIN = c(20010811, 20060101, 20130101),
#                         END = c(20051231, 20160110, 20160109))

# gw <- handle_gsod_old(action = "download_weather",
#                   location = "724828_93241",
#                   time_interval = c(2010, 2012),
#                   station_list = stat_list,
#                   quiet = TRUE)

# weather <- handle_gsod_old(gw, add.DATE = FALSE)[[1]]$weather

# make_chill_plot(tempResponse(stack_hourly_temps(fix_weather(weather)),
#                              Start_JDay = 300, End_JDay = 50),
#                 "Chill_Portions", start_year = 2010,
#                 end_year = 2012, metriclabel = "Chill Portions",
#                 misstolerance = 50)
```

---

handle_ucipm *List, download or convert to chillR format data from the UCIPM database*

Description

This function can do three things related to the University of California Integrated Pest Management (UCIPM) database: 1. it can list stations that are close to a specified position (geographic coordinates) 2. it can retrieve weather data for a named weather station 3. it can 'clean' downloaded data, so that they can easily be used in chillR Which of these functions is carried out depends on the action argument.

Usage

```
handle_ucipm(
  action,
  location = NA,
  time_interval = NA,
  station_list = california_stations,
  stations_to_choose_from = 25,
  drop_most = TRUE,
  end_at_present = TRUE
)
```

Arguments

action	if this is the character string "list_stations", the function will return a list of the weather stations from the database that are closest to the geographic coordinates specified by location. if this is the character string "download_weather", the function will attempt to download weather data from the database for the station named by the location argument, which should then be a character string corresponding to the chillRcode of the station (which you can get by running this function in 'list_stations mode) if this is a downloaded weather file (downloaded by running this function in 'download weather' mode), the function cleans the file and makes it ready for use in chillR. If the input is just a dataframe (not a list, as produced with this function), you have to specify the database name with the database argument
location	either a vector of geographic coordinates (for the 'list_stations' mode), or the 'chillRcode' of a weather station in the specified database (for the 'download_weather' mode. When running this function for data cleaning only, this is not needed.
time_interval	numeric vector with two elements, specifying the start and end date of the period of interest. Only required when running in 'list_stations' or 'download weather' mode
station_list	if the list of weather stations has already been downloaded, the list can be passed to the function through this argument. This can save a bit of time, since it can take a bit of time to download the list, which can have several MB.
stations_to_choose_from	if the location is specified by geographic coordinates, this argument determines the number of nearby stations in the list that is returned.
drop_most	boolean variable indicating if most columns should be dropped from the file. If set to TRUE (default), only essential columns for running chillR functions are retained.

`end_at_present` boolean variable indicating whether the interval of interest should end on the present day, rather than extending until the end of the year specified under time_interval[2] (if time_interval[2] is the current year).

Details

This function can run independently, but it is also called by the get_weather and weather2chillR functions, which some users might find a bit easier to handle.

the UCIPM dataset is described here: http://ipm.ucdavis.edu/WEATHER/

under the 'list_stations' mode, several formats are possible for specifying the location vector, which can consist of either two or three coordinates (it can include elevation). Possible formats include c(1,2,3), c(1,2), c(x=1,y=2,z=3), c(lat=2,long=1,elev=3). If elements of the vector are not names, they are interpreted as c(Longitude, Latitude, Elevation).

The 'chillRCode' is generated by this function, when it is run with geographic coordinates as location inputs. In the list of nearby stations that is returned then, the chillRCode is provided and can then be used as input for running the function in 'downloading' mode. For downloading the data, use the same call as before but replace the location argument with the chillRCode.

Value

The output depends on the action argument. If it is 'list_stations', the function returns a list of station_to_choose_from weather stations that are close to the specified location. This list also contains information about how far away these stations are (in km), how much the elevation difference is (if elevation is specified; in m) and how much overlap there is between the data contained in the database and the time period specified by time_interval. If action is 'download_weather' the output is a list of two elements: 1. database="CIMIS" 2. the downloaded weather record, extended to the full duration of the specified time interval. If action is a weather data.frame or a weather record downloaded with this function (in 'download_weather' mode), the output is the same data in a format that is easy to use in chillR. If drop_most was set to TRUE, most columns are dropped.

Note

Many databases have data quality flags, which may sometimes indicate that data aren't reliable. These are not considered by this function!

The station list provided by the UC IPM database doesn't contain geographic positions of the stations, which can only be accessed by station-specific websites. This function will access this information only if it was not given on the website in early 2016. Station information based on a download at that time is stored in the california_station dataset included in chillR. This was done to reduce the run time for the handle_ucipm function. It will probably be okay for the foreseeable future (stations don't change very quickly). A new version of this table can be produces with the make_california_UCIPM_station_list() function.

Author(s)

Eike Luedeling

References

The chillR package:

Luedeling E, Kunz A and Blanke M, 2013. Identification of chilling and heat requirements of cherry trees - a statistical approach. International Journal of Biometeorology 57,679-689.

Examples

```
# All examples are disabled, because the database is sometimes unavailable. This then generates
# an error when R runs its package functionality checks. To run the examples, remove the # mark,
# before running the code.
#
#handle_ucipm(action="list_stations",location=c(x=-122,y=38.5),time_interval=c(2012,2012))
#gw<-handle_ucipm(action="download_weather",location="WINTERS.A",time_interval=c(2012,2012))
#weather<-handle_ucipm(gw)$weather
#make_chill_plot(tempResponse(stack_hourly_temps(fix_weather(weather)),Start_JDay=300,End_JDay=50),
#           "Chill_Portions",start_year=2010,end_year=2012,metriclabel="Chill Portions",
#                misstolerance = 50)
```

---

identify_common_string

Identify shared leading or trailing character strings

Description

Compares all elements of a vector of numbers or character strings and returns TRUE if they are all the same, FALSE otherwise.

Usage

```
identify_common_string(strings, leading = TRUE)
```

Arguments

strings	vector of strings to be evaluated.
leading	boolean variable indicating whether the function should look for common strings at the beginning (leading==TRUE) or end (leading==FALSE) of the strings. Default is TRUE.

Value

if there is a leading (if leading==TRUE) or trailing (if leading==FALSE) string that all elements of strings have in common, this string is returned; NA otherwise.

Author(s)

Eike Luedeling

Examples

```
identify_common_string(c("Temp_01","Temp_02","Temp_03"))
identify_common_string(c("Temp_01","Temp_02","Temp_03"),leading=FALSE)
identify_common_string(c("file1.csv","file2.csv","file3.csv"),leading=FALSE)
```

`interpolate_gaps` *Linear gap interpolation*

Description

This function linearly interpolates gaps in data series, such as daily temperature records.

Usage

```
interpolate_gaps(x)
```

Arguments

`x`	a numeric vector, or a vector that can be coerced with as.numeric. Missing values are either NA or non-numeric values.

Details

The function returns a list with two elements: interp is a new vector, in which all gaps in x have been linearly interpolated. missing is a second vector, which contains information on which values were filled in by interpolation.

Value

`interp`	numeric vector, in which all gaps in x have been linearly interpolated
`missing`	boolean vector of the same length as interp and x, which marks all gaps in x as TRUE

Author(s)

Eike Luedeling

References

Luedeling E, Kunz A and Blanke M, 2013. Identification of chilling and heat requirements of cherry trees - a statistical approach. International Journal of Biometeorology 57,679-689.

Examples

```
weather<-make_all_day_table(KA_weather)
Tmin_int<-interpolate_gaps(KA_weather[,"Tmin"])
weather[,"Tmin"]<-Tmin_int$interp
weather[,"Tmin_interpolated"]<-Tmin_int$missing

Tmax_int<-interpolate_gaps(KA_weather[,"Tmax"])
weather[,"Tmax"]<-Tmax_int$interp
weather[,"Tmax_interpolated"]<-Tmax_int$missing

#this function is integrated into the fix_weather function, but it can also be run on its own.
```

---

interpolate_gaps_hourly

Interpolate gaps in hourly temperature records

Description

Using idealized temperature curves for guidance, this function interpolated hourly temperature data.

Usage

```
interpolate_gaps_hourly(
  hourtemps,
  latitude = 50,
  daily_temps = NULL,
  interpolate_remaining = TRUE,
  return_extremes = FALSE,
  minimum_values_for_solving = 5,
  runn_mean_test_length = 5,
  runn_mean_test_diff = 5,
  daily_patch_max_mean_bias = NA,
  daily_patch_max_stdev_bias = NA
)
```

Arguments

hourtemps	data.frame containing hourly temperatures. This has to contain columns c("Year","Month","Day","Hour",
latitude	the geographic latitude (in decimal degrees) of the location of interest
daily_temps	list of (chillR compliant) daily temperature data sets for patching gaps in the record.
interpolate_remaining	boolean parameter indicating whether gaps remaining after the daily record has been patched (or after solving temperature equations, if (daily_temps==NULL)) should be linearly interpolated.

return_extremes
: boolean parameters indicating whether daily minimum and maximum temperatures used for the interpolation should be part of the output table. Defaults to FALSE.

minimum_values_for_solving
: integer specifying the minimum number of hourly temperature values that must be available for the solving function to be applied. Must be greater than 1 (otherwise you get an error). Since according to the idealized temperature curves used here, a given daily extreme temperature is related to hourly temperatures of about a 12-hour period, values above 12 are not useful. Note that relatively large numbers for this parameter raise the reliability of the interpolated values, but they restrict the number of missing values in a day, for which the procedure produces results.

runn_mean_test_length
: integer specifying the length of the period, for which a running mean test for is applied to daily records after the solving procedure. This aims to remove spurious values that can sometimes arise during solving. This test checks for all daily minimum and maximum temperature values, if they differ from the mean of the surrounding values by more than runn_mean_test_diff. If this is the case, they are set to NA, and have to be filled by other means (from proxy data or by interpolation). Defaults to 5, which means each value is compared to the mean of the 2 previous and 2 following days.

runn_mean_test_diff
: integer specifying the maximum tolerable difference between solved daily extreme temperature values and the mean for the surrounding days. See description of runn_mean_test_length for more details. Defaults to 5.

daily_patch_max_mean_bias
: maximum acceptable mean difference between the daily extreme temperatures of daily temperature records used as proxy and daily extreme temperatures in the dataset that is to be interpolated. If the bias between stations is greater than this, the station is not considered a useful proxy and not used for filling gaps.

daily_patch_max_stdev_bias
: maximum acceptable standard deviation of the difference between the daily extreme temperatures of daily temperature records used as proxy and daily extreme temperatures in the dataset that is to be interpolated. If the bias between stations is greater than this, the station is not considered a useful proxy and not used for filling gaps.

Details

Many agroclimatic metrics are calculated from hourly temperature data. chillR provides functions for generating hourly data from daily records, which are often available. Small gaps in such daily records can easily be closed through linear interpolation, with relatively small errors, so that complete hourly records can be generated. However, many sites have recorded actual hourly temperatures, which allow much more accurate site-specific assessments. Such records quite often have gaps, which need to be closed before calculating most agroclimatic metrics (such as Chill Portions). Linear interpolation is not a good option for this, because daily temperature curves are not linear. Moreover, when gaps exceed a certain number of hours, important featured would be missed (e.g.

interpolating between temperatures at 8 pm and 8 am may miss all the cool hours of the day, which would greatly distort chill estimates).

This function solves this problem by using an idealized daily temperature curve as guide to the interpolation of hourly temperature data.

These are the steps: 1) produce an idealized temperature curve for the site (which requires site latitude as an input), assuming minimum and maximum temperatures of 0 and 1 degrees C, respectively. The calculations are based on equations published by Spencer (1971), Almorox et al. (2005) and Linvill (1990, though I modified these slightly to produce a smooth curve). This curve describes the expected relationship of the temperature for the respective hour with minimum and maximum temperatures of the same, previous or next day (depending on the time of day), according to idealized temperature curve. At this point, however, these daily minimum or maximum temperatures aren't known yet.

2) determine minimum and maximum temperatures for each day. For each minimum and maximum daily temperature, the expected relationships between hourly temperatures and daily extremes determined in step 1, combined with the hourly temperatures that were observed can be interpreted as an overdetermined set of equations that define these temperatures. Since few days will follow the ideal curve precisely, and there are usually more than two equations that define the same daily temperature extreme value, these equations can only be solved numerically. This is implemented with the qr.solve function, which can provide estimates of the minimum and maximum temperatures for all days from the available hourly records.

3) interpolate gaps in the record of estimated daily temperature extremes. There can be days, when the number of recorded hourly temperatures isn't sufficient for inferring daily minimum or maximum temperatures. The resulting gaps are closed by linear interpolation (this may produce poor results if gaps are really large, but this isn't currently addressed).

4) compute an idealized daily temperature curve for all days, based on estimated daily temperature extremes (using the make_hourly_temperatures function).

5) calculate deviation of recorded temperatures from idealized curve.

6) linearly interpolate deviation values using the interpolate_gaps function.

7) add interpolated deviation values to idealized temperature curve.

Value

data frame containing interpolated temperatures for all hours within the interval defined by the first and last day of the hourtemps input.

Author(s)

Eike Luedeling

References

Linvill DE, 1990. Calculating chilling hours and chill units from daily maximum and minimum temperature observations. HortScience 25(1), 14-16.

Spencer JW, 1971. Fourier series representation of the position of the Sun. Search 2(5), 172.

Almorox J, Hontoria C and Benito M, 2005. Statistical validation of daylength definitions for estimation of global solar radiation in Toledo, Spain. Energy Conversion and Management 46(9-10), 1465-1471)

Examples

```
Winters_gaps<-make_JDay(Winters_hours_gaps[1:2000,])
colnames(Winters_gaps)[5:6]<-c("Temp","original_Temp")
interp<-interpolate_gaps_hourly(hourtemps=Winters_gaps,latitude=38.5)

#plot results: interpolated temperatures are shown in red, measured temperatures in black.
plot(interp$weather$Temp[1:120]~c(interp$weather$JDay[1:120]+
   interp$weather$Hour[1:120]/24),type="l",
   col="RED",lwd=2,xlab="JDay",ylab="Temperature")
lines(interp$weather$Temp_measured[1:120]~c(interp$weather$JDay[1:120]+
   interp$weather$Hour[1:120]/24),lwd=2)
```

JDay_count *Count days between two Julian dates*

Description

This function counts the days between two Julian dates, taking into account whether the season extends past the end of a calender year and whether the count is to be done for a leap year.

Usage

```
JDay_count(start_date, end_date, season = NA, leap_year = FALSE)
```

Arguments

start_date	integer ranging from 1 to 366, indicating a Julian date. This is the start date of the interval of interest.
end_date	integer ranging from 1 to 366, indicating a Julian date. This is the end date of the interval of interest.
season	integer vector of length 2, specifying the beginning and end of the phenology season, respectivcely. If this is not specified, the start_date and end_date are used to define the season.
leap_year	either a Boolean parameter indicating whether the count should be done for a leap year, or an integer specyfing the year, for which the calculation is to be done. The function then determines automatically, whether this is a leap year.

Value

Boolean result (TRUE/FALSE) of the comparison.

Author(s)

Eike Luedeling

Examples

```
JDay_count(start_date=320,end_date=20,season=c(305,59),leap_year=2004)
```

`JDay_earlier` *Check whether a Julian date is before or after another one*

Description

For two Julian dates, this function checks whether the first date is earlier than the second date within a user-defined phenological season. This is particularly useful for seasons that start in one year and end in the next, because simple > or < operations can produce wrong results then.

Usage

```
JDay_earlier(check_date, ref_date, season = c(1, 366))
```

Arguments

`check_date`	integer ranging from 1 to 366, indicating a Julian date. This is the date for which to check whether it is before the reference date. If this is a vector, all elements are checked against the reference date.
`ref_date`	integer ranging from 1 to 366, indicating a Julian date. This is the reference date.
`season`	integer vector of length 2, specifying the beginning and end of the phenology season, respectivcely.

Value

Boolean result (TRUE/FALSE) of the comparison.

Author(s)

Eike Luedeling

Examples

```
JDay_earlier(check_date=10,ref_date=365,season=c(305,59))
```

JDay_later *Check whether a Julian date is after another one*

Description

For two Julian dates, this function checks whether the first date is later than the second date within a user-defined phenological season. This is particularly useful for seasons that start in one year and end in the next, because simple > or < operations can produce wrong results then.

Usage

```
JDay_later(check_date, ref_date, season = c(1, 366))
```

Arguments

check_date	integer ranging from 1 to 366, indicating a Julian date. This is the date for which to check whether it is after the reference date. If this is a vector, all elements are checked against the reference date.
ref_date	integer ranging from 1 to 366, indicating a Julian date. This is the reference date.
season	integer vector of length 2, specifying the beginning and end of the phenology season, respectivcely.

Value

Boolean result (TRUE/FALSE) of the comparison.

Author(s)

Eike Luedeling

Examples

```
JDay_later(check_date=10,ref_date=365,season=c(305,59))
```

KA_bloom *Cherry bloom data for Klein-Altendorf, Germany*

Description

Bloom data of sweet cherry var. 'Schneiders spaete Knorpelkirsche' recorded at Klein-Altendorf, Germany, the experimental station of the University of Bonn

Format

A data frame with the following 2 variables.

Year a numeric vector, indicating the observation year

pheno a vector that, when coerced by as.numeric, contains bloom data in Julian dates (day of the year)

Source

data were collected by Achim Kunz and Michael Blanke, University of Bonn

References

Luedeling E, Kunz A and Blanke M, 2013. Identification of chilling and heat requirements of cherry trees - a statistical approach. International Journal of Biometeorology 57,679-689.

Examples

```
data(KA_bloom)
```

KA_weather *Weather data for Klein-Altendorf, Germany*

Description

Daily temperature data from Klein-Altendorf, Germany, for use in combination with the example phenology dataset KA_bloom.

Format

A data frame with observations on the following 5 variables.

Year a numeric vector - the observation year

Month a numeric vector - the observation month

Day a numeric vector - the observation day

Tmax a numeric vector - daily maximum temperature

Tmin a numeric vector - daily minimum temperature

Source

data were collected by Achim Kunz and Michael Blanke, University of Bonn

References

Luedeling E, Kunz A and Blanke M, 2013. Identification of chilling and heat requirements of cherry trees - a statistical approach. International Journal of Biometeorology 57,679-689.

Examples

```
data(KA_weather)
```

`leap_year` *Leap year finder*

Description

This function determines whether a given year is a leap year

Usage

```
leap_year(x)
```

Arguments

`x`	integer value, representing year number

Details

Takes a year number as input, and returns TRUE if this is a leap year, and FALSE if not

Value

boolean variable (TRUE or FALSE)

Author(s)

Eike Luedeling, but based on pseudocode from Wikipedia

References

https://en.wikipedia.org/wiki/Leap_year

Examples

```
leap_year(2015)
leap_year(2016)
```

load_ClimateWizard_scenarios *Load climate wizard scenarios*

Description

This is a slightly modified version of the load_temperature_scenarios function that can load climate scenarios downloaded with the getClimateWizardData and saved with the save_temperature_scenarios function. This separate function is necessary, because the climate scenarios are expressed as lists, with one element being a data.frame.

Usage

```
load_ClimateWizard_scenarios(path, prefix)
```

Arguments

path	character string indicating the file path where the files are to be written.
prefix	character string specifying the prefix for all files.

Value

a list of temperature scenarios.

Author(s)

Eike Luedeling

Examples

```
temps<-list(Element1=data.frame(a=1,b=2),Element2=data.frame(a=c(2,3),b=c(8,4)))
# save_temperature_scenarios(temps,path=getwd(),prefix="temperatures")
# temps_reloaded<-load_temperature_scenarios(path=getwd(),prefix="temperatures")
```

load_temperature_scenarios *Load temperature scenarios*

Description

The temperature_generation can produce synthetic temperature scenarios, but it can take a while to run, especially for large ensembles of climate scenarios. The save_temperature_scenarios function can then save these scenarios to disk as a series of .csv files, so that they can later be used again, without re-running the generation function. Conversely, the load_temperature_scenarios function allows reading the data back into R. This function also works with any other list of data.frames.

Usage

```
load_temperature_scenarios(path, prefix)
```

Arguments

path	character string indicating the file path where the files are to be written.
prefix	character string specifying the prefix for all files.

Value

a list of temperature scenarios.

Author(s)

Eike Luedeling

Examples

```
temps<-list(Element1=data.frame(a=1,b=2),Element2=data.frame(a=c(2,3),b=c(8,4)))
# save_temperature_scenarios(temps,path=getwd(),prefix="temperatures")
# temps_reloaded<-load_temperature_scenarios(path=getwd(),prefix="temperatures")
```

---

make_all_day_table *Fill in missing days in incomplete time series*

Description

Time series often have gaps, and these are often not marked by 'no data' values but simply missing from the dataset. This function completes the time series by adding lines for all these missing records. For these lines, all values are set to 'NA'. By setting timestep<-"hour", this function can also process hourly data. Where data are provided at a time resolution that is finer than timestep, values are aggregated (by calculating the mean) to timestep resolution (e.g. when data are at 15-minute resolution, they will be aggregated to hourly average values - at timestep=="hour" - or daily average values - at timestep=="day").

Usage

```
make_all_day_table(
  tab,
  timestep = "day",
  input_timestep = timestep,
  tz = "GMT",
  add.DATE = TRUE,
  no_variable_check = FALSE,
  aggregation_hours = NULL
)
```

Arguments

tab	a data.frame containing a time series dataset. It should have columns c("Year", "Month", "Day") or c("YEAR", "MONTH","DAY") or "YEARMODA".
timestep	time step for the table. This defaults to 'day' but can also be 'hour'
input_timestep	can also be 'day' or 'hour' and defaults to the value assigned to timestep. If timestep is 'day' and input_timestep is 'hour', hourly records are aggregated to daily Tmin, Tmean and Tmax.
tz	timezone. Defaults to GMT. While it isn't important in what time zone the temperatures were recorded, the onset of daylight savings time can cause problems. 'GMT' is the correct setting in cases were the recorded times weren't adjusted according to daylight savings time (i.e. no hours omitted or double-counted because of such adjustment).
add.DATE	boolean parameter indicating whether a column called DATE which contains the IOSdate should be added to the output data.frame.
no_variable_check	boolean parameter to indicate whether the function should check if the dataset contains the usual chillR temperature variables. Defaults to TRUE, but should be set to FALSE for different data formats.
aggregation_hours	vector or list consisting of three integers that specify how the function should search for daily minimum and maximum temperatures in hourly datasets, when not all hourly temperatures have been observed. This is only relevant during conversion from hourly to daily data. Tmin and Tmax can only be derived when temperatures have been recorded during the coldest and warmest parts of the day, respectively. The function should therefore check if records are available for these times. The elements of 'aggregation_hours' describe window sizes for the times (as number of hours), during which the coldest and warmest temperature typically occurs. The first two elements (which can be named 'min_hours' and 'max_hours') specify the number of hours contained in these windows for the cold and warm parts of the day, respectively. These hours are determined by computing mean hourly temperatures over the entire weather record, disaggregated by month to account for the impact of daylength. The third element, 'hours_needed' specifies how many records during these windows have to have been recorded. 'aggregation_hours' defaults to NULL, in which case the parameter is ignored.

Value

data frame containing all the columns of the input data frame, but one row for each day between the start and end of the dataset. Data values for the missing rows are filled in as 'NA'. Dates are expressed as c("YEARMODA","DATE","Year","Month","Day"). In this, 'DATE' is the date in ISOdate format.

Author(s)

Eike Luedeling

References

Luedeling E, Kunz A and Blanke M, 2013. Identification of chilling and heat requirements of cherry trees - a statistical approach. International Journal of Biometeorology 57,679-689.

Examples

```
#fill in missing lines in a weather dataset (modified from KA_weather)
day_to_day<-make_all_day_table(KA_weather[c(1:10,20:30),],timestep="day")

#fill in missing hours in the Winters_hours_gaps dataset
Winters_hours<-subset(Winters_hours_gaps, select = -c(Temp_gaps))[1:2000,]
hour_to_hour<-make_all_day_table(Winters_hours,timestep="hour",input_timestep="hour")

#convert Winters_hours_gaps dataset into daily temperature data (min, max, mean)
hour_to_day<-make_all_day_table(Winters_hours,timestep="day",input_timestep="hour")
hour_to_day<-make_all_day_table(Winters_hours,timestep="day",input_timestep="hour",
                                aggregation_hours=c(3,3,2))
```

---

make_california_UCIPM_station_list

Makes a list of the UC IPM weather stations

Description

Makes a list of the weather stations contained in the UC IPM database, with geographic coordinates. This requires parsing through quite a few websites, because the coordinates don't seem to be stored in one central (and easily accessible) place. Hence this is much slower than one might expect. A shortcut is the california_stations dataset supplied with chillR, which contains the result of running this function in February 2016. The default in the other relevant functions will be the use of this pre-stored list, but if the current station coverage is needed, this function can help. Having said this, station coverage probably won't change very rapidly, so in most cases, the california_stations dataset should be enough.

Usage

```
make_california_UCIPM_station_list()
```

Value

a data.frame containing stations from the California UC IPM database (), with the following columns: "Name", "Code", "Interval", "Lat", "Long", "Elev".

Author(s)

Eike Luedeling

References

The chillR package:

Luedeling E, Kunz A and Blanke M, 2013. Identification of chilling and heat requirements of cherry trees - a statistical approach. International Journal of Biometeorology 57,679-689.

Examples

```
#cali_stats<-make_california_UCIPM_station_list()
```

`make_chill_plot` *Plot climate metrics over time*

Description

This function generates a plot of a climate metric over multiple years, including an indication of data quality, i.e. the share of missing values. Output can be either an R plot or a .png image

Usage

```
make_chill_plot(
  chill,
  model,
  start_year = NA,
  end_year = NA,
  metriclabel = NULL,
  yearlabel = "End_year",
  misstolerance = 10,
  image_type = NA,
  outpath = NA,
  filename = NA,
  fonttype = "serif",
  plotylim = NA,
  plottitle = NULL
)
```

Arguments

`chill`	a chill object generated either with the chilling function or with tempResponse. For this function to work properly, the chill object should have been subjected to quality control (i.e. metrics should have been calculated from weather records with a QC element. If you prepare weather data with fix_weather, this should work.)
`model`	the name of the column of the chill object that contains the metric to be displayed
`start_year`	the first year shown in the diagram. Default to NA, which means the first date on record is used as start_year.

end_year	the last year shown in the diagram. Default to NA, which means the last date on record is used as end_year.
metriclabel	character string that can be used for labeling the y-axis in the plot. If this is not specified, the function will use the model argument.
yearlabel	character string indicating the name of the column in the chill object that is to be used for the time axis.
misstolerance	Percentage of missing values that leads to exclusion of an annual value from plotting.
image_type	Character string indicating the file format that should be output. Image files are only produced for the moment, if this is "png". All other values, as well as the default NA lead to output as an R plot only.
outpath	Path to the folder where the images should be saved. Should include a trailing "/".
filename	Suffix of the filenames for output graph files. These will be amended by the name of the metric and by the file extension.
fonttype	The type of font to be used for the figures. Can be 'serif' (default) for a Times New Roman like font, 'sans' for an Arial type font or 'mono' for a typewriter type font.
plotylim	numeric vector of length 2 indicating the extent of the y axis. Defaults to NA, which means that y limits are determined automatically.
plottitle	character string indicating the plot title. Defaults to NULL for no title.

Details

Plots climatic metrics computed with chilling or tempResponse, indicating the completeness of the temperature record by shades of gray.

Value

only a side effect - plot of climate metric over time; bars are color coded according to the number of missing values. Bars with numbers of missing values above the misstolerance are not show and instead marked '*' (to distinguish them from 0 counts)

Author(s)

Eike Luedeling

Examples

```
make_chill_plot(tempResponse(stack_hourly_temps(fix_weather(KA_weather[KA_weather$Year>2005,]))),
"Chill_Portions",start_year=1990,end_year=2010,metriclabel="Chill Portions")
```

`make_climate_scenario` *Make climate scenario*

Description

Function to make climate scenarios for plotting from a list of climate metric data, e.g. produced by `tempResponse_daily_list`.

Usage

```
make_climate_scenario(
  metric_summary,
  caption = NULL,
  labels = names(metric_summary),
  time_series = FALSE,
  historic_data = NULL,
  add_to = NULL
)
```

Arguments

`metric_summary`	character string specifying the folder holding the files, from which the scenario is to be built.
`caption`	vector of up to three character strings indicating the caption to be displayed in the respective plot panel; the elements of this vector are displayed on different lines. If `caption_above == TRUE` in `plot_climate_scenario`, only the first element is displayed.
`labels`	numeric vector containing labels for the scenarios. This defaults to the names of elements in `metric_summary`.
`time_series`	Boolean, indicating if the scenario contains a time series.
`historic_data`	a data.frame containing a dataset of historic observations that is similar in structure to metric_summary (should have column indicating the year and the metric to be plotted, with identical names to `metric_summary`). Defaults to `NULL`, which means that no historic data is included.
`add_to`	list of climate scenarios that the newly created one is to be added to.

Value

a list of climate scenario objects, which can be supplied to plot_climate_scenarios.

Author(s)

Eike Luedeling

Examples

```
chill<-chilling(stack_hourly_temps(fix_weather(KA_weather[which(KA_weather$Year>1990),]),
   latitude=50.4))
multi_chills<-list('2001'=chill,'2005'=chill,'2009'=chill)
chills_to_plot<-make_climate_scenario(multi_chills,caption=c("Historic","data"),
   time_series=TRUE,historic_data=chill)
chills_to_plot<-make_climate_scenario(multi_chills,caption=c("Future1"),add_to=chills_to_plot)
chills_to_plot<-make_climate_scenario(multi_chills,caption=c("Future2"),add_to=chills_to_plot)
plot_climate_scenarios(chills_to_plot,metric="Chill_portions",metric_label="Chill Portions")
```

---

make_climate_scenario_from_files

Make climate scenario from multiple saved csv files

Description

Many climate scenarios we may want to plot consist of data stored across many files. These files typically contain certain character strings that mark, e.g. the RCP scenario or the point in time. This function facilitates accessing such files by allowing the specification of search string (criteria_list), according to which files are selected. They are then converted into climate_scenario files that can become part of a list passed to plot_climate_scenarios for plotting.

Usage

```
make_climate_scenario_from_files(
  metric_folder,
  criteria_list,
  caption = NULL,
  time_series = FALSE,
  labels = NULL,
  historic_data = NULL
)
```

Arguments

metric_folder	character string specifying the folder holding the files, from which the scenario is to be built.
criteria_list	list of character vectors that specify parts of the file names that are common to all files of a particular scenario. These can be single strings or vectors of string. In the latter case, occurrence of either of the elements in a file name is sufficient. The selection criteria are applied iteratively, i.e. first all files containing the first element of 'criteria_list' are selected, then those containing the second element, and so forth.

caption	vector of up to three character strings indicating the caption to be displayed in the respective plot panel; the elements of this vector are displayed on different lines. If caption_above==TRUE in plot_climate_scenario, only the first element is displayed.
time_series	Boolean, indicating if the scenario contains a time series.
labels	numeric vector containing labels for the time scenarios - only used for time series.
historic_data	a data.frame containing at least two columns named the same as 'metric' and 'year_name'.

Value

a climate scenario object, which can be part of a list supplied to plot_climate_scenarios.

Author(s)

Eike Luedeling

Examples

```
# historic_scenario<-make_climate_scenario(metric_folder=chillout_folder,
#                                          criteria_list=list(cult,c(1975,2000,2015)),
#                                          caption=c("Historic","data"),
#                                         time_series=TRUE,
#                                         labels=c(1975,2000,2015),
#                                         historic_data=historic_data)
```

---

make_daily_chill_figures

Produce image of daily chill and heat accumulation

Description

Function to make figures showing the mean rate of chill and heat accumulation for each day of the year, as well as as the standard deviation.

Usage

```
make_daily_chill_figures(
  daily_chill,
  file_path,
  models = c("Chilling_Hours", "Utah_Chill_Units", "Chill_Portions", "GDH"),
  labels = NA
)
```

Arguments

daily_chill	a daily chill object. This should be generated with the daily_chill function.
file_path	the path where data should be saved. Can either end with '/' or include a prefix for all images that are produced.
models	column names of the data.frame stored in daily_chill's daily_chill object that contain the metrics to be plotted. Defaults to four standard metrics of interest in fruit tree phenology analysis.
labels	labels to be used in the plots for the metrics listed under models. This defaults to NA, which means that the character strings given in models are used for the figures. If alternative labels are to be used, these should be given as a vector of length length(models).

Details

Chill metrics are calculated as given in the references below. Chilling Hours are all hours with temperatures between 0 and 7.2 degrees C. Units of the Utah Model are calculated as suggested by Richardson et al. (1974) (different weights for different temperature ranges, and negation of chilling by warm temperatures). Chill Portions are calculated according to Fishman et al. (1987a,b). More honestly, they are calculated according to an Excel sheet produced by Amnon Erez and colleagues, which converts the complex equations in the Fishman papers into relatively simple Excel functions. These were translated into R. References to papers that include the full functions are given below. Growing Degree Hours are calculated according to Anderson et al. (1986), using the default values they suggest. This function uses the Kendall package.

Value

data frame containing all information used to make the figures that are saved. For each Julian Date, means and standard deviations of all chill and heat metrics are saved. In addition, Mann-Kendall tests are performed for daily accumulations of all metrics. p and tau values from this test indicate the level of statistical significance. This non-parametric test is reliable for time series data.

Note

After doing extensive model comparisons, and reviewing a lot of relevant literature, I do not recommend using the Chilling Hours or Utah Models, especially in warm climates! The Dynamic Model (Chill Portions), though far from perfect, seems much more reliable.

Author(s)

Eike Luedeling

References

Model references:

Chilling Hours:

Weinberger JH (1950) Chilling requirements of peach varieties. Proc Am Soc Hortic Sci 56, 122-128

Bennett JP (1949) Temperature and bud rest period. Calif Agric 3 (11), 9+12

Utah Model:

Richardson EA, Seeley SD, Walker DR (1974) A model for estimating the completion of rest for Redhaven and Elberta peach trees. HortScience 9(4), 331-332

Dynamic Model:

Erez A, Fishman S, Linsley-Noakes GC, Allan P (1990) The dynamic model for rest completion in peach buds. Acta Hortic 276, 165-174

Fishman S, Erez A, Couvillon GA (1987a) The temperature dependence of dormancy breaking in plants - computer simulation of processes studied under controlled temperatures. J Theor Biol 126(3), 309-321

Fishman S, Erez A, Couvillon GA (1987b) The temperature dependence of dormancy breaking in plants - mathematical analysis of a two-step model involving a cooperative transition. J Theor Biol 124(4), 473-483

Growing Degree Hours:

Anderson JL, Richardson EA, Kesner CD (1986) Validation of chill unit and flower bud phenology models for 'Montmorency' sour cherry. Acta Hortic 184, 71-78

Model comparisons and model equations:

Luedeling E, Zhang M, Luedeling V and Girvetz EH, 2009. Sensitivity of winter chill models for fruit and nut trees to climatic changes expected in California's Central Valley. Agriculture, Ecosystems and Environment 133, 23-31

Luedeling E, Zhang M, McGranahan G and Leslie C, 2009. Validation of winter chill models using historic records of walnut phenology. Agricultural and Forest Meteorology 149, 1854-1864

Luedeling E and Brown PH, 2011. A global analysis of the comparability of winter chill models for fruit and nut trees. International Journal of Biometeorology 55, 411-421

Luedeling E, Kunz A and Blanke M, 2011. Mehr Chilling fuer Obstbaeume in waermeren Wintern? (More winter chill for fruit trees in warmer winters?). Erwerbs-Obstbau 53, 145-155

Review on chilling models in a climate change context:

Luedeling E, 2012. Climate change impacts on winter chill for temperate fruit and nut production: a review. Scientia Horticulturae 144, 218-229

The PLS method is described here:

Luedeling E and Gassner A, 2012. Partial Least Squares Regression for analyzing walnut phenology in California. Agricultural and Forest Meteorology 158, 43-52.

Wold S (1995) PLS for multivariate linear modeling. In: van der Waterbeemd H (ed) Chemometric methods in molecular design: methods and principles in medicinal chemistry, vol 2. Chemie, Weinheim, pp 195-218.

Wold S, Sjostrom M, Eriksson L (2001) PLS-regression: a basic tool of chemometrics. Chemometr Intell Lab 58(2), 109-130.

Mevik B-H, Wehrens R, Liland KH (2011) PLS: Partial Least Squares and Principal Component Regression. R package version 2.3-0. http://CRAN.R-project.org/package0pls.

Some applications of the PLS procedure:

Luedeling E, Kunz A and Blanke M, 2013. Identification of chilling and heat requirements of cherry trees - a statistical approach. International Journal of Biometeorology 57,679-689.

Yu H, Luedeling E and Xu J, 2010. Stronger winter than spring warming delays spring phenology on the Tibetan Plateau. Proceedings of the National Academy of Sciences (PNAS) 107 (51), 22151-22156.

Yu H, Xu J, Okuto E and Luedeling E, 2012. Seasonal Response of Grasslands to Climate Change on the Tibetan Plateau. PLoS ONE 7(11), e49230.

The exact procedure was used here:

Luedeling E, Guo L, Dai J, Leslie C, Blanke M, 2013. Differential responses of trees to temperature variation during the chilling and forcing phases. Agricultural and Forest Meteorology 181, 33-42.

The chillR package:

Luedeling E, Kunz A and Blanke M, 2013. Identification of chilling and heat requirements of cherry trees - a statistical approach. International Journal of Biometeorology 57,679-689.

Examples

```
weather<-fix_weather(KA_weather[which(KA_weather$Year>2005),])

dc<-daily_chill(stack_hourly_temps(weather,50.4), 11,models=list(Chill_Portions=Dynamic_Model))

# md<-make_daily_chill_figures(dc, paste(getwd(),"/daily_chill_",sep=""),models="Chill_Portions",
#  labels="Chill Portions")
```

`make_daily_chill_plot` *Plot daily climate metric accumulation throughout the year*

Description

This function generates a plot of the accumulation of a climate metric throughout the year. Its standard output are the mean daily accumulation and the standard deviation. It is also possible to add one or several so-called focusyears to add the daily accumulation during these years to the plots. Plots can be produced in R or directly exported as .png files.

Usage

```
make_daily_chill_plot(
  daily_chill,
  metrics = NA,
  startdate = 1,
  enddate = 366,
  useyears = NA,
  metriclabels = NA,
  focusyears = "none",
  cumulative = FALSE,
  image_type = NA,
  outpath = NA,
```

```
  filename = NA,
  fonttype = "serif",
  title = NA,
  plotylim = NA
)
```

Arguments

daily_chill	a daily chill object generated with the daily_chill function, which can calculate several standard chilling metrics or be supplied with user-written temperature models. Since the format for the input file must meet certain requirements, I recommend that you follow the steps shown in the example below to prepare it.
metrics	list of the metrics to be evaluated. This defaults to NA, in which case the function makes a guess on what metrics you want to calculated. This is done by choosing all column headers that are not required for a daily_chill object.
startdate	the first day of the season for which the metrics are to be summarized (as a Julian date = day of the year)
enddate	the last day of the season for which the metrics are to be summarized (as a Julian date = day of the year)
useyears	if only certain years are to be used, these can be provided here as a numeric vector. Defaults to NA, which means all years in the daily_chill object are used.
metriclabels	Character vector with labels for each metric to be analyzed. Defaults to NA, which means that the strings passed as metrics will be used.
focusyears	Numeric vector containing the years that are to be highlighted in the plot. Years for which no data are available are automatically removed.
cumulative	Boolean argument (TRUE or FALSE) indicating whether the climate metric should be shown as daily accumulation rates or as cumulative accumulation.
image_type	Character string indicating the file format that should be output. Image files are only produced for the moment, if this is "png". All other values, as well as the default NA lead to output as an R plot only.
outpath	Path to the folder where the images should be saved. Should include a trailing "/". The folder must already exists.
filename	Suffix of the filenames for output graph files. These will be amended by the name of the metric and by the file extension.
fonttype	The type of font to be used for the figures. Can be 'serif' (default) for a Times New Roman like font, 'sans' for an Arial type font or 'mono' for a typewriter type font.
title	title of the plot (if unhappy with the default).
plotylim	numeric vector of length 2 indicating the extent of the y axis. Defaults to NA, which means that y limits are determined automatically.

Details

Plots daily accumulation of climatic metrics, such as winter chill, as daily accumulation rates or as cumulative accumulation. A legend is only added, when focusyears are also shown. Otherwise the plot is reasonably self-explanatory.

Value

The main purpose of the function is a side effect - plots of daily climate metric accumulation. However, all the data used for making the plots is returned as a list containing an element for each metric, which consists of a data.table with the daily means, standard deviation and daily values for all focusyears.

Author(s)

Eike Luedeling

Examples

```
day_chill<-make_daily_chill_plot(daily_chill(stack_hourly_temps(fix_weather(
  KA_weather[which(KA_weather$Year>2005),])),
  running_mean=11),focusyears=c(2001,2005),cumulative=TRUE,startdate=300,enddate=30)
```

---

make_daily_chill_plot2

Plot daily climate metric accumulation throughout the year (2)

Description

This function generates a plot of the accumulation of a climate metric throughout the year. Its standard output are the mean daily accumulation and the standard deviation. It is also possible to add one or several so-called focusyears to add the daily accumulation during these years to the plots. Plots can be produced in R or directly exported as .png files.

Usage

```
make_daily_chill_plot2(
  daily,
  metrics = NA,
  startdate = 1,
  enddate = 366,
  useyears = NA,
  metriclabels = NA,
  focusyears = "none",
  cumulative = FALSE,
  fix_leap = TRUE
)
```

Arguments

daily	an object generated with the daily_chill function, which can calculate several standard chilling metrics or be supplied with user-written temperature models. Since the format for the input file must meet certain requirements, I recommend that you follow the steps shown in the example below to prepare it.
metrics	list of the metrics to be evaluated. This defaults to NA, in which case the function makes a guess on what metrics you want to calculated. This is done by choosing all column headers that are not required for a daily_chill object.
startdate	the first day of the season for which the metrics are to be summarized (as a Julian date = day of the year)
enddate	the last day of the season for which the metrics are to be summarized (as a Julian date = day of the year)
useyears	if only certain years are to be used, these can be provided here as a numeric vector. Defaults to NA, which means all years in the daily_chill object are used.
metriclabels	Character vector with labels for each metric to be analyzed. Defaults to NA, which means that the strings passed as metrics will be used.
focusyears	Numeric vector containing the years that are to be highlighted in the plot. Years for which no data are available are automatically removed.
cumulative	Boolean argument (TRUE or FALSE) indicating whether the climate metric should be shown as daily accumulation rates or as cumulative accumulation.
fix_leap	boolean parameter indicating whether the anomaly that can originate when leaf years are present in the data should be smoothed by interpolating between Dec 30 and Jan 1 in leap years.

Details

Plots daily accumulation of climatic metrics, such as winter chill, as daily accumulation rates or as cumulative accumulation. A legend is only added, when focusyears are also shown. Otherwise the plot is reasonably self-explanatory.

Value

The main purpose of the function is a side effect - plots of daily climate metric accumulation. However, all the data used for making the plots is returned as a list containing an element for each metric, which consists of a data.table with the daily means, standard deviation and daily values for all focusyears.

Author(s)

Eike Luedeling

Examples

```
daily<-daily_chill(stack_hourly_temps(fix_weather(
  KA_weather[which(KA_weather$Year>2005),])),running_mean=11)
```

```
make_daily_chill_plot2(daily,metrics=c("Chill_Portions","GDH"),cumulative=TRUE,
   startdate=300,enddate=30,focusyears=c(2009,2008))
```

`make_hourly_temps` *Make hourly temperature record from daily data*

Description

This function generates hourly temperature records for a particular location from daily minimum and maximum temperatures and latitude.

Usage

```
make_hourly_temps(latitude, year_file, keep_sunrise_sunset = FALSE)
```

Arguments

`latitude`	the geographic latitude (in decimal degrees) of the location of interest
`year_file`	a data frame containing data on daily minimum temperature (called Tmin), daily maximum temperature (called Tmax), and date information. Dates can either be specified by two columns called Year and JDay, which contain the Year and Julian date (day of the year), or as three columns called Year, Month and Day. year_file cannot have any missing values, so it may be a good idea to process the relevant columns with make_all_day_table and interpolate_gaps before.
`keep_sunrise_sunset`	boolean variable indicating whether information on sunrise, sunset and daylength, which is calculated for producing hourly temperature records, should be preserved in the output. Defaults to FALSE.

Details

Temperature estimates are based on an idealized daily temperature curve that uses a sine curve for daytime warming and a logarithmic decay function for nighttime cooling. The input data frame can have more columns, which are preserved, but ignored in the processing. References to papers outlining the procedures are given below.

Note that this function should be able to generate hourly temperatures for all latitudes, but it uses an algorithm designed for locations with regular day/night behavior. It may therefore be that the curves aren't very realistic for very short or very long days, or especially for polar days and nights.

Value

data frame containing all the columns of year_file, plus 24 columns for hourly temperatures (called Hour_1 ... Hour_24).

Author(s)

Eike Luedeling

References

Luedeling E, Kunz A and Blanke M, 2013. Identification of chilling and heat requirements of cherry trees - a statistical approach. International Journal of Biometeorology 57,679-689.

Luedeling E, Girvetz EH, Semenov MA and Brown PH, 2011. Climate change affects winter chill for temperate fruit and nut trees. PLoS ONE 6(5), e20155.

The temperature interpolation is described in

Linvill DE, 1990. Calculating chilling hours and chill units from daily maximum and minimum temperature observations. HortScience 25(1), 14-16.

Calculation of sunrise, sunset and daylength was done according to

Spencer JW, 1971. Fourier series representation of the position of the Sun. Search 2(5), 172.

Almorox J, Hontoria C and Benito M, 2005. Statistical validation of daylength definitions for estimation of global solar radiation in Toledo, Spain. Energy Conversion and Management 46(9-10), 1465-1471)

Examples

```
weather<-fix_weather(KA_weather)

THourly<-make_hourly_temps(50.4,weather$weather)

#in most cases, you're probably better served by stack_hour_temperatures
```

---

make_JDay *Make Julian Day in dataframe*

Description

This function produced Julian Dates (days of the year) from columns "Day", "Month" and "Year" in a dataframe.

Usage

```
make_JDay(dateframe)
```

Arguments

dateframe	data.frame, which should contain date information specified as columns "Day", "Month" and "Year"

Value

Returns the same data.frame, but with column "JDay" added. This then contains the Julian Dates.

Author(s)

Eike Luedeling

References

The chillR package:

Examples

```
dates<-data.frame(Year=c(1977,1980,2004,2011,2016),Month=c(11,8,3,12,8),Day=c(1,21,2,24,2))
make_JDay(dates)
```

---

make_multi_pheno_trend_plot

Combine multiple phenology contour plots in one figure

Description

For multiple datasets, this function plots surface plots relating mean temperatures during specified periods to annually recurring variables (e.g. flowering). It produces one panel per dataset and plots them all in one figure. Plots can be produced in R or directly exported as .png files.

Usage

```
make_multi_pheno_trend_plot(
  pheno_list,
  fixed_weather,
  split_month = 6,
  outpath = NA,
  file_name = NA,
  image_type = "png",
  fonttype = "serif",
  percol = 5,
  xlabel = NA,
  ylabel = NA,
  height_factor = 0.8
)
```

Arguments

pheno_list	a data.frame with the following columns: varieties (contains a character string), Start_chill (the start of the chill period, in Julian days), End_chill (the end of the chill period, in Julian days), Start_heat (the start of the forcing period, in Julian days), End_heat (the end of the forcing period, in Julian days), Link (the complete path to a csv file that contains all the annual observations for the dataset, with columns Year and pheno)
fixed_weather	daily weather, as produced with the fix_weather function
split_month	the month after which to start a new season. Defaults to 6, meaning the new season will start in July.
outpath	Path to the folder where the images should be saved. Should include a trailing "/". The folder must already exists.
file_name	name of the image file to be produced, if image_type='png'.
image_type	Character string indicating the file format that should be output. Image files are only produced for the moment, if this is "png". All other values, as well as the default NA lead to output as an R plot only.
fonttype	The type of font to be used for the figures. Can be 'serif' (default) for a Times New Roman like font, 'sans' for an Arial type font or 'mono' for a typewriter type font.
percol	number of plots to be placed in a column.
xlabel	label for the x-axis (if unhappy with the default).
ylabel	label for the y-axis (if unhappy with the default).
height_factor	height of the resulting png figure (if this is a png) relative to the width of the plot (e.g. 1 or 0.7, defaults to 0.8).

Details

This function is only useful, if you want to plot several surface plots in the same figure. These must relate to the same weather dataset. Arguably, this function isn't quite ready to be released, but it performs some useful functions that you may be interested in...

Value

Only a side effect is produced: either a .png file or an R graphic showing the multi-panel contour figure.

Author(s)

Eike Luedeling

Examples

```
#this example uses arbitrarily modified versions of the KA_bloom dataset, and the starts
#end ends of the periods are also arbitraty. So the outputs may not make a lot of sense...

weather<-fix_weather(KA_weather[which(KA_weather$Year>2000),])
```

```
pheno_list<-data.frame(varieties=c("KA1","KA2","KA3","KA4"), Start_chill=c(270,305,315,320),
  End_chill=c(15,20,35,40), Start_heat=c(17,25,40,45),End_heat=c(90,100,110,115),
  Link=c("KA1.csv","KA2.csv","KA3.csv","KA4.csv"))

 # write.csv(KA_bloom,"KA1.csv",row.names=FALSE)
 KA_bloom$pheno<-as.numeric(as.character(KA_bloom$pheno))+10
 # write.csv(KA_bloom,"KA2.csv",row.names=FALSE)
 KA_bloom$pheno<-KA_bloom$pheno+10
 # write.csv(KA_bloom,"KA3.csv",row.names=FALSE)
 KA_bloom$pheno<-KA_bloom$pheno+10
 # write.csv(KA_bloom,"KA4.csv",row.names=FALSE)

# make_multi_pheno_trend_plot(pheno_list,weather, split_month=6,
#            outpath=NA,file_name=NA,image_type="",fonttype="serif",percol=2)
```

---

make_pheno_trend_plot *Make image showing phenology response to temperatures during two phases*

Description

The timing of many developmental stages of temperate plants is understood to depend on temperatures during two phases. This function seeks to illustrate this dependency by plotting phenological dates as colored surface, as a function of mean temperatures during both phases, which are indicated on the x and y axes.

Usage

```
make_pheno_trend_plot(
  weather_data_frame,
  split_month = 6,
  pheno,
  use_Tmean = FALSE,
  Start_JDay_chill,
  End_JDay_chill,
  Start_JDay_heat,
  End_JDay_heat,
  outpath,
  file_name,
  plot_title,
  ylabel = NA,
  xlabel = NA,
  legend_label = NA,
  image_type = "png",
  colorscheme = "normal",
  fonttype = "serif"
)
```

Arguments

weather_data_frame	a dataframe containing daily minimum and maximum temperature data (in columns called Tmin and Tmax, respectively), and/or mean daily temperature (in a column called Tmean). There also has to be a column for Year and one for JDay (the Julian date, or day of the year). Alternatively, the date can also be given in three columns (Year, Month and Day).
split_month	the procedure analyzes data by phenological year, which can start and end in any month during the calendar year (currently only at the beginning of a month). This variable indicates the last month (e.g. 5 for May) that should be included in the record for a given phenological year. All subsequent months are assigned to the following phenological year.
pheno	a data frame that contains information on the timing of phenological events by year. It should consist of two columns called Year and pheno. Data in the pheno column should be in Julian date (day of the year).
use_Tmean	boolean variable indicating whether or not the column Tmean from the weather_data_frame should be used as input for the PLS analysis. If this is set to FALSE, Tmean is calculated as the arithmetic mean of Tmin and Tmax.
Start_JDay_chill	the Julian date, on which the first relevant period (e.g. the chilling phase) starts
End_JDay_chill	the Julian date, on which the first relevant period (e.g. the chilling phase) ends
Start_JDay_heat	the Julian date, on which the second relevant period (e.g. the forcing phase) starts
End_JDay_heat	the Julian date, on which the second relevant period (e.g. the forcing phase) ends
outpath	the output path
file_name	the output file name
plot_title	the title of the plot
ylabel	the label for the y-axis. There is a default, but in many cases, it may be desirable to customize this
xlabel	the label for the x-axis. There is a default, but in many cases, it may be desirable to customize this
legend_label	the label for the legend (color scheme). There is a default, but in many cases, it may be desirable to customize this
image_type	the type of image to produce. This currently has only two options: "tiff" or anything else (the default). If this is not "tiff", a png image is produced. The "tiff" option was added to produce publishable figures that adhere to the requirements of most scientific journals.
colorscheme	the color scheme for the figure. This currently has only two options: "bw" or anythings else (the default). "bw" produces a grayscale image, otherwise the figure will be in color
fonttype	font style to be used for the figure. Can be 'serif' (default) or 'sans'.

Details

The generation of the color surface is based on the Kriging technique, which is typically used for interpolation of spatial data. The use for this particular purpose is a bit experimental. The function uses the Krig function from the fields package.

Value

pheno	data frame containing all data needed for reproducing the plot: Year (during which the phenological event occurred - the year in which the phenological season indicated by split_month ended), pheno (the date on which the phenological event occurred), Chill_Tmean (mean temperature during the first relevant phase), Heat_Tmean (mean temperature during the second relevant phase) and Year_Tmean (mean annual temperature - not actually used in the plot)
ylabel	character string used for labeling the y axis
xlabel	character string used for labeling the x axis

Author(s)

Eike Luedeling

References

Guo L, Dai J, Wang M, Xu J, Luedeling E, 2015. Responses of spring phenology in temperate zone trees to climate warming: a case study of apricot flowering in China. Agricultural and Forest Meteorology 201, 1-7.

Guo L, Dai J, Ranjitkar S, Xu J, Luedeling E, 2013. Response of chestnut phenology in China to climate variation and change. Agricultural and Forest Meteorology 180, 164-172.

Luedeling E, Guo L, Dai J, Leslie C, Blanke M, 2013. Differential responses of trees to temperature variation during the chilling and forcing phases. Agricultural and Forest Meteorology 181, 33-42.

the interpolation was done according to:

Furrer, R., Nychka, D. and Sain, S., 2012. Fields: Tools for spatial data. R package version 6.7.

Reference to the chillR package:

Luedeling E, Kunz A and Blanke M, 2013. Identification of chilling and heat requirements of cherry trees - a statistical approach. International Journal of Biometeorology 57,679-689.

Examples

```
weather<-fix_weather(KA_weather)

#the output of the PLS function (PLS_pheno, plotted with plot_PLS) can be used to select
#phases that are likely relevant for plant phase timing. See respective examples for running
#these functions.

file_path<-paste(getwd(),"/",sep="")

#mpt<-make_pheno_trend_plot(weather_data_frame = weather$weather, split_month = 6,
#      pheno = KA_bloom, use_Tmean = FALSE, Start_JDay_chill = 260,
```

```
#      End_JDay_chill = 64, Start_JDay_heat = 44, End_JDay_heat = 103,
#      outpath = file_path, file_name = "pheno_trend_plot",
#      plot_title = "Impacts of chilling and forcing temperatures on cherry phenology",
#      ylabel = NA, xlabel = NA, legend_label = NA, image_type = "png",
#      colorscheme = "normal")
```

`ordered_climate_list` *Sort files in a folder, so that numbers are in ascending sequence*

Description

Sometimes lists of strings that contain numbers aren't listed automatically in the sequence we would expect, e.g. because numbers below ten are lacking leading zeros (as in c("a1","a10","a100","a11"...)). This function recognizes all shared leading and trailing symbols around the numeric part of such strings and sorts the list according to the embedded numbers.

Usage

```
ordered_climate_list(strings, file_extension = NA)
```

Arguments

`strings`	vector of character strings to be sorted according to embedded numbers.
`file_extension`	character string specifying the extension of the file type to be selected. This can also be any other trailing string that marks all vector elements to be selected. This isn't required for the function to run, but may be necessary if the string list of interest contains, for instance, different file types, of which you only want to work with one.

Value

subset of the strings vector that only contains the elements that end on file_extension and are sorted in ascending order according to the numeric parts of the strings.

Author(s)

Eike Luedeling

Examples

```
ordered_climate_list(c("Temp1_ws30.csv","Temp1_ws30.xls",
                       "Temp10_ws30.csv","Temp10_ws30.xls",
                       "Temp2_ws30.csv","Temp2_ws30.xls"),"csv")
ordered_climate_list(c("Tx12", "Tx2","Tx4","Tx1"))
```

patch_daily_temperatures *Patch gaps in daily weather records*

Description

This function is deprecated. Better use patch_daily_temps!

Usage

```
patch_daily_temperatures(
  weather,
  patch_weather,
  vars = c("Tmin", "Tmax"),
  max_mean_bias = NA,
  max_stdev_bias = NA
)
```

Arguments

weather	chillR-compatible weather record to be patched
patch_weather	list of chillR-compatible weather records to be used for patching holes in weather. They are used sequentially, until all have been used or until there are no holes left.
vars	vector of column names to be considered in patching. Defaults to c("Tmin","Tmax"), the most common variables in chillR applications.
max_mean_bias	maximum mean bias of auxiliary data compared to the original dataset (applied to all variables in vars). If this threshold is exceeded, the respective variable from that particular dataset will not be used. Defaults to NA, meaning no records are excluded.
max_stdev_bias	maximum standard deviation of the bias in the auxiliary data compared to the original dataset (applied to all variables in vars). If this threshold is exceeded, the respective variable from that particular dataset will not be used. Defaults to NA, meaning no records are excluded.

Details

This function uses auxiliary data sources to fill gaps in daily weather data. It can accommodate multiple sources of auxiliary information, which are used in the user-specified sequence. There have to be some overlapping records for this to work, because without bias correction, this procedure could produce erroneous records. Bias correction is done by computing the mean difference between main and auxiliary data for each variable and adjusting for it in filling the gaps. You can specify a maximum mean bias and a maximum standard deviation of the bias to exclude unsuitable records that aren't similar enough to the original data.

Value

list of two elements: weather (the patched weather record, with additional columns specifying the data source for each value) and statistics (containing data.frames for each element of patch_weather that indicate the mean bias, the number of values that were filled from this source and the number of missing records that remained after exhausting this auxiliary data source.)

Author(s)

Eike Luedeling

Examples

```
gap_weather<-KA_weather[1:100,]
gap_weather[c(3,4,7:15,20,22:25,27:28,35:45,55,67,70:75,80:88,95:97),"Tmin"]<-NA
gap_weather[c(10:25,30,36:44,50,57,65,70:80,86,91:94),"Tmax"]<-NA
p1<-KA_weather[65:95,]
p1$Tmin<-p1$Tmin-2
p2<-KA_weather[c(15:40,60:80),]
p2$Tmax<-p2$Tmax+3
p3<-KA_weather[12:35,]
p3$Tmax<-p3$Tmax-2
p4<-KA_weather
p4$Tmax<-p4$Tmax+0.5
patch_weather<-list(stat1=p1,st2=p2,home=p3,last=p4)

patched<-patch_daily_temperatures(gap_weather,patch_weather,max_mean_bias=1)
```

---

patch_daily_temps *Patch gaps in daily weather records - updated*

Description

This is the successor function of patch_daily_temperatures, which will no longer be updated.

Usage

```
patch_daily_temps(
  weather,
  patch_weather,
  vars = c("Tmin", "Tmax"),
  max_mean_bias = NA,
  max_stdev_bias = NA,
  time_interval = "month"
)
```

Arguments

weather	chillR-compatible weather record to be patched
patch_weather	list of chillR-compatible weather records to be used for patching holes in weather. They are used sequentially, until all have been used or until there are no holes left.
vars	vector of column names to be considered in patching. Defaults to c("Tmin","Tmax"), the most common variables in chillR applications.
max_mean_bias	maximum mean bias of auxiliary data compared to the original dataset (applied to all variables in vars). If this threshold is exceeded, the respective variable from that particular dataset will not be used. Defaults to NA, meaning no records are excluded.
max_stdev_bias	maximum standard deviation of the bias in the auxiliary data compared to the original dataset (applied to all variables in vars). If this threshold is exceeded, the respective variable from that particular dataset will not be used. Defaults to NA, meaning no records are excluded.
time_interval	time interval for which mean bias and standard deviation of the bias are to be evaluated. This defaults to "month", which means that the function looks for overlapping days between weather and patch_weather for each calendar month. Bias correction is then also done on a monthly basis. 'time_interval' can also assume other values, such as 'week' or '2 weeks'.

Details

The patch_daily_temps function uses auxiliary data sources to fill gaps in daily weather data. It can accommodate multiple sources of auxiliary information, which are used in the user-specified sequence. There have to be some overlapping records for this to work, because without bias correction, this procedure could produce erroneous records. Bias correction is done by computing the mean difference between main and auxiliary data for each variable and adjusting for it in filling the gaps. You can specify a maximum mean bias and a maximum standard deviation of the bias to exclude unsuitable records that aren't similar enough to the original data. When patching records, the function breaks the calendar year down into smaller intervals that can be specified with the 'time_interval' parameter (this was not possible in [chillR::patch_daily_temperatures], but is recommended for accurate results).

Value

list of two elements: weather (the patched weather record, with additional columns specifying the data source for each value) and statistics (containing data.frames for each element of patch_weather that indicate the mean bias, the number of values that were filled from this source and the number of missing records that remained after exhausting this auxiliary data source.)

Author(s)

Eike Luedeling

Examples

```
gap_weather<-KA_weather[1:100,]
gap_weather[c(3,4,7:15,20,22:25,27:28,35:45,55,67,70:75,80:88,95:97),"Tmin"]<-NA
gap_weather[c(10:25,30,36:44,50,57,65,70:80,86,91:94),"Tmax"]<-NA
p1<-KA_weather[65:95,]
p1$Tmin<-p1$Tmin-2
p2<-KA_weather[c(15:40,60:80),]
p2$Tmax<-p2$Tmax+3
p3<-KA_weather[12:35,]
p3$Tmax<-p3$Tmax-2
p4<-KA_weather
p4$Tmax<-p4$Tmax+0.5
patch_weather<-list(stat1=p1,st2=p2,home=p3,last=p4)

patch_daily_temps(gap_weather,patch_weather)

patch_daily_temps(gap_weather,patch_weather,max_mean_bias=0.1,time_interval="2 weeks")
```

PhenoFlex *PhenoFlex*

Description

Combined model of the dynamic model for chill accumulation and the GDH model

Usage

```
PhenoFlex(
  temp,
  times,
  A0 = 6319.5,
  A1 = 5.939917e+13,
  E0 = 3372.8,
  E1 = 9900.3,
  slope = 1.6,
  Tf = 4,
  s1 = 0.5,
  Tu = 25,
  Tb = 4,
  Tc = 36,
  yc = 40,
  Delta = 4,
  Imodel = 0L,
  zc = 190,
  stopatzc = TRUE,
  deg_celsius = TRUE,
```

```
  basic_output = TRUE
)
```

Arguments

temp	Vector of temperatures.
times	numeric vector. Optional times at which the temperatures where measured, if not given, hourly temperatures will be assumed
A0	numeric. Parameter A_0 of the dynamic model
A1	numeric. Parameter A_1 of the dynamic model
E0	numeric. Parameter E_0 of the dynamic model
E1	numeric. Parameter E_1 of the dynamic model
slope	numeric. Slope parameter for sigmoidal function
Tf	numeric. Transition temperature (in degree Kelvin) for the sigmoidal function
s1	numeric. Slope of transition from chill to heat accumulation
Tu	numeric. GDH optimal temperature
Tb	numeric. GDH base temperature (lower threshold)
Tc	numeric. GDH upper temperature (upper threshold)
yc	numeric. Critical value defining end of chill accumulation
Delta	numeric. Width of Gaussian heat accumulation model
Imodel	integer. Heat accumulation model: 0 for GDH and 1 for Gaussian
zc	numeric. Critical value of z determining the end of heat accumulation
stopatzc	boolean. If 'TRUE', the PhenoFlex is applied until the end of the temperature series. Default is to stop once the value zc has been reached.
deg_celsius	boolean. whether or not the temperature vector and the model temperature parameters are in degree Celsius (Kelvin otherwise)
basic_output	boolean. If 'TRUE', only the bloomindex is returned as a named element of the return list.

Value

A list is returned with named element 'bloomindex', which is the index at which blooming occurs. When 'basic_output=FALSE' also 'x', 'y', 'z' and 'xs' are returned as named element of this list, which are numeric vectors of the same length as the input vector 'temp' containing the hourly temperatures.

Author(s)

Carsten Urbach <urbach@hiskp.uni-bonn.de>

Examples

```
data(KA_weather)
hourtemps <- stack_hourly_temps(KA_weather, latitude=50.4)
iSeason <- genSeason(hourtemps, years=c(2009))
zc <- 190
yc <- 40
x <- PhenoFlex(temp=hourtemps$hourtemps$Temp[iSeason[[1]]],
               times=c(1: length(hourtemps$hourtemps$Temp[iSeason[[1]]])),
               zc=zc, stopatzc=TRUE, yc=yc, basic_output=FALSE)
DBreakDay <- x$bloomindex
ii <- c(1:DBreakDay)
plot(x=ii, y=x$z[ii], xlab="Hour Index", ylab="z", col="red", type="l")
abline(h=zc, lty=2)
plot(x=ii, y=x$y[ii], xlab="Hour Index", ylab="y", col="red", type="l")
abline(h=yc, lty=2)
```

PhenoFlex_fixedDynModelGAUSSwrapper

PhenoFlex_fixedDynModelGAUSSwrapper

Description

PhenoFlex wrapper function for the 'phenologyFitter' function using the GAUSS heat accumulation model and parameters of the dynamical model fixed.

Usage

```
PhenoFlex_fixedDynModelGAUSSwrapper(
  x,
  par,
  A0 = 139500,
  A1 = 2.567e+18,
  E0 = 4153.5,
  E1 = 12888.8,
  slope = 1.6,
  Tf = 4
)
```

Arguments

x	data.frame with at least columns 'Temp' and 'JDay'
par	numeric vector of length 11 with the 'PhenoFlex' fit parameters in the following order: 1. yc, 2. zc, 3. s1, 4. Tu, 5. E0, 6. E1, 7, A0, 8. A1, 9. Tf, 10. Tc, 11. Tb and 12. slope. For details see PhenoFlex
A0	numeric. Parameter A_0 of the dynamic model
A1	numeric. Parameter A_1 of the dynamic model
E0	numeric. Parameter E_0 of the dynamic model

E1	numeric. Parameter E_1 of the dynamic model
slope	numeric. Slope parameter for sigmoidal function
Tf	numeric. Transition temperature (in degree Kelvin) for the sigmoidal function

Value

A single numeric value with the JDay prediction for the temperaturs in 'x$Temp' and the PhenoFlex parameters in 'par'.

PhenoFlex_fixedDynModelwrapper

PhenoFlex_fixedDynModelwrapper

Description

PhenoFlex wrapper function for the 'phenologyFitter' function using the GDH heat accumulation model and parameters of the dynamical model fixed. The default values for the dynamic model parameters are from the excel file with unknown origin.

Usage

```
PhenoFlex_fixedDynModelwrapper(
  x,
  par,
  A0 = 139500,
  A1 = 2.567e+18,
  E0 = 4153.5,
  E1 = 12888.8,
  slope = 1.6,
  Tf = 4
)
```

Arguments

x	data.frame with at least columns 'Temp' and 'JDay'
par	numeric vector of length 11 with the 'PhenoFlex' fit parameters in the following order: 1. yc, 2. zc, 3. s1, 4. Tu, 5. E0, 6. E1, 7, A0, 8. A1, 9. Tf, 10. Tc, 11. Tb and 12. slope. For details see PhenoFlex
A0	numeric. Parameter A_0 of the dynamic model
A1	numeric. Parameter A_1 of the dynamic model
E0	numeric. Parameter E_0 of the dynamic model
E1	numeric. Parameter E_1 of the dynamic model
slope	numeric. Slope parameter for sigmoidal function
Tf	numeric. Transition temperature for the sigmoidal function

Value

A single numeric value with the JDay prediction for the temperaturs in ‘x$Temp‘ and the PhenoFlex parameters in ‘par‘.

`PhenoFlex_GAUSSwrapper` *PhenoFlex_GAUSSwrapper*

Description

PhenoFlex wrapper function for the ‘phenologyFitter‘ function using the Gaussian heat accumulation model

Usage

```
PhenoFlex_GAUSSwrapper(x, par)
```

Arguments

`x`	data.frame with at least columns ‘Temp‘ and ‘JDay‘
`par`	numeric vector of length 11 with the ‘PhenoFlex‘ fit parameters in the following order: 1. yc, 2. zc, 3. s1, 4. Tu, 5. E0, 6. E1, 7, A0, 8. A1, 9. Tf, 10. Delta, 11. s. For details see PhenoFlex

Value

A single numeric value with the JDay prediction for the temperatures in ‘x$Temp‘ and the PhenoFlex parameters in ‘par‘.

`PhenoFlex_GDHwrapper` *PhenoFlex_GDHwrapper*

Description

PhenoFlex wrapper function for the ‘phenologyFitter‘ function using the GDH heat accumulation model

Usage

```
PhenoFlex_GDHwrapper(x, par)
```

Arguments

`x`	data.frame with at least columns ‘Temp‘ and ‘JDay‘
`par`	numeric vector of length 11 with the ‘PhenoFlex‘ fit parameters in the following order: 1. yc, 2. zc, 3. s1, 4. Tu, 5. E0, 6. E1, 7, A0, 8. A1, 9. Tf, 10. Tc, 11. Tb and 12. slope. For details see PhenoFlex

Value

A single numeric value with the JDay prediction for the temperaturs in ‘x$Temp‘ and the PhenoFlex parameters in ‘par‘.

`phenologyFit` *phenologyFit*

Description

Constructor for class ‘phenologyFit‘

Usage

```
phenologyFit()
```

Value

an empty object of class ‘phenologyFit‘. It contains the named elements ‘model_fit‘ with the returned object from GenSA, ‘par‘ the best fit parameters, ‘pbloomJDays‘ the predicted bloom JDays and the inputs ‘par.guess‘, ‘modelfn‘, ‘bloomJDays‘, and ‘SeasonList‘. They are all set to ‘NULL‘ by this function.

`phenologyFitter` *phenologyFitter*

Description

phenologyFitter

Usage

```
phenologyFitter(
  par.guess = NULL,
  modelfn = PhenoFlex_GDHwrapper,
  bloomJDays,
  SeasonList,
 control = list(smooth = FALSE, verbose = TRUE, maxit = 1000, nb.stop.improvement = 250),
  lower,
  upper,
  seed = 1235433,
  ...
)
```

Arguments

par.guess	numeric vector. Initial guesses for fit parameters. This can be set to 'NULL', in which case 'GenSA' choses initial parameters.
modelfn	function. Model function which computes the index in 'temperatures' at which blooming occures. It must have as first argument a data frame with at least the two columns 'Temp' and 'JDays' for one season, see 'SeasonList'. It can have further arguments which can be passed via '...'. The 'modelfn' must return a single numeric value for the predicted bloom JDay for that season. 'NA' is an allowed return value if no blooming occures in that season. The default is the PhenoFlex with GDH as heat accumulation. Alternative is PhenoFlex_GAUSSwrapper with GAUSSian heat accumulation. But this function can also be user defined.
bloomJDays	integer vector. vector of observed bloom JDays per year
SeasonList	list. Must be a list of data frames, each data frame for one season. Each data.frame must at least have a column 'Temp' with the temperature vector and 'JDays' with the corresponding JDay vector. Can be generated by e.g. genSeasonList. 'length(SeasonList)' must be equal to 'length(bloomJDays)'.
control	control parameters to 'GenSA', see 'GenSA::GenSA'
lower	Vector with length of 'par.guess'. Lower bounds for components.
upper	Vector with length of 'par.guess'. Upper bounds for components.
seed	integer seed for the random number generator used by 'GenSA'.
...	further parameters to be passed on to 'modelfn'.

Value

an object of class 'phenologyFit'. It contains the named elements 'model_fit' with the returned object from GenSA, 'par' the best fit parameters, 'pbloomJDays' the predicted bloom JDays and the inputs 'par.guess', 'modelfn', 'bloomJDays', 'lower', 'upper', 'control', 'SeasonList' and '...'.

Author(s)

Carsten Urbach <urbach@hiskp.uni-bonn.de>

Examples

```
## this example does not make sense as a fit, but demonstrates
## how to use `phenologyFitter`
data(KA_weather)
data(KA_bloom)
hourtemps <- stack_hourly_temps(KA_weather, latitude=50.4)
SeasonList <- genSeasonList(hourtemps$hourtemps, years=c(2007,2008))
par <- c(40, 190, 0.5, 25, 3372.8, 9900.3, 6319.5, 5.939917e13, 4, 36, 4, 1.6)
upper <- c(41, 200, 1, 30, 4000, 10000, 7000, 6.e13, 10, 40, 10, 50)
lower <- c(38, 180, 0.1, 0, 3000, 9000, 6000, 5.e13, 0, 0, 0, 0.05)
X <- phenologyFitter(par.guess=par, bloomJDays=KA_bloom$pheno[c(24,25)],
  SeasonList=SeasonList, lower=lower, upper=upper,
  control=list(smooth=FALSE, verbose=TRUE, maxit=10, nb.stop.improvement=5))
```

```
summary(X)
plot(X)
```

`plot.bootstrap_phenologyFit` *plot bootstrap_phenologyFit*

Description

Generic function to plot a ‘bootstrap_phenologyFit‘ object

Usage

```
## S3 method for class 'bootstrap_phenologyFit'
plot(
  x,
  ylim = c(0.9 * min(c(x$object$bloomJDays, x$object$pbloomJDays)), 1.1 *
    max(c(x$object$bloomJDays, x$object$pbloomJDays))),
  ...
)
```

Arguments

`x`	object of class ‘bootstrap_phenologyFit‘ to plot.
`ylim`	numeric vector of length 2 with the limit for the y-axis
`...`	additional graphical parameters to pass on.

Value

No return value.

`plot.phenologyFit` *plot phenologyFit*

Description

Generic function to plot a ‘phenologyFit‘ object

Usage

```
## S3 method for class 'phenologyFit'
plot(
  x,
  ylim = c(0.9 * min(c(x$bloomJDays, x$pbloomJDays)), 1.1 * max(c(x$bloomJDays,
    x$pbloomJDays))),
  ...
)
```

Arguments

x	object of class ‘phenologyFit‘ to plot.
ylim	numeric vector of length 2 with the limit for the y-axis
...	additional graphical parameters to pass on.

Value

No return value.

---

plot_climateWizard_scenarios

Plot mutltiple ClimateWizard scenarios obtained with getClimateWizard_scenarios

---

Description

This function plots multiple scenarios obtained with the getClimateWizard_scenarios function.

Usage

```
plot_climateWizard_scenarios(
  getscenarios_element,
  low_filter = -1000,
  high_filter = 1000,
  color = "cadetblue"
)
```

Arguments

getscenarios_element	outputs from the getClimateWizard_scenarios function
low_filter	numeric value specifying the lowest plausible value for the variable of interest. This is sometimes necessary to exclude erroneous values in the ClimateWizard database.
high_filter	numeric value specifying the highest plausible value for the variable of interest. This is sometimes necessary to exclude erroneous values in the ClimateWizard database.
color	color to be used for the plots.

Value

returns nothing, but a plot is produced as a side effect.

Author(s)

Eike Luedeling

References

Girvetz E, Ramirez-Villegas J, Navarro C, Rodriguez C, Tarapues J, undated. ClimateWizard REST API for querying climate change data. https://github.com/CIAT-DAPA/climate_wizard_api

Examples

```
#example is #d out, because of runtime issues.
#getC<-getClimateWizard_scenarios(coordinates=c(longitude=6.99,latitude=50.62),
#                                  scenarios=c("rcp45","rcp45","rcp85","rcp85"),
#                                  start_years=c(2035,2070,2035,2070),
#                                  end_years=c(2065,2100,2065,2100),
#                                  metric=c("monthly_tmean"),
#                                  GCMs=c("all"))
#plot_climateWizard_scenarios(getC,low_filter=-6,high_filter=6,color="red")
```

---

plot_climate_scenarios

Plot multiple chilling scenario groups (or for other metrics)

Description

For quantifying climate risks, it is useful to generate many version of plausible weather for particular climate scenarios. This can, for example, be done with the temperature_generation function. This function facilitates illustration of these results by providing various options to show them as boxplots. The function can plot either a single panel of climate scenarios or multiple panels side by side.

Usage

```
plot_climate_scenarios(
  climate_scenario_list,
  metric,
  metric_label,
  year_name = "End_year",
  label_sides = "both",
  ylim = c(0, NA),
  reference_line = NULL,
  col_rect = NA,
  col_line = NA,
  hist_col = NA,
  texcex = 2,
  caption_above = FALSE,
  family = "serif",
  no_scenario_numbers = FALSE
)
```

Arguments

climate_scenario_list	list of lists containing information about the chill scenarios. These lists must have an element named 'data' which should contain a data.frame with a column named the same as the 'metric' argument, which contains (numeric) information to be plotted. Additional optional elements are 'time_series' (Boolean, indicating if a time series is to be plotted), 'labels' (vector of length 'length(data)' containing labels for the scenarios - if this is a time series scenario, these must be numeric; if the data are not a time series, the labels aren't shown in the plot, because there wouldn't normally be enough space - only numbers are shown there, and the legend is provided in the value returned by this function), 'caption' (up to three character strings indicating the caption to be displayed in the respective plot panel; the elements of this vector are displayed on different lines. If caption_above==TRUE, only the first element is displayed) and 'historic_data' (a data.frame containing at least two columns named the same as 'metric' and 'year_name'). documentation of 'make_chill_scenario_plot' for details on these.
metric	character string corresponding to the name of the column that contains the data of interest in the climate_scenario_list data.frames (and if applicable the historic_data data.frame).
metric_label	character string specifying the y-axis label.
year_name	character string indicating the name of the time column in the historic_data data.frame.
label_sides	indicates what sides of the plot y-axis labels are to be drawn. Can be "left', "right" or "both". If label_sides assumes any other value, no labels are plotted.
ylim	numeric vector of length 2, specifying the lower and upper limits of the y-axis. If either of these two values is NA, it is automatically selected based on the data range.
reference_line	numeric vector of length 1, 2 or 3, specifying a horizontal reference bar to be drawn across the plot (e.g. to indicate exceedance of a threshold). A reference_line argument of length 1 is interpreted by drawing a line across the plot at the specified value. If length(reference_line)==2, the values are interpreted as lower and upper limit of a rectangular threshold area. If length(reference_line)==3, the lowest and highest values are used to draw a rectangle and the median value to draw a line (e.g. to show a best estimate and a confidence interval around it).
col_rect	color code or name for the color of the reference_line rectangle.
col_line	color code or name for the color of the reference_line line.
hist_col	color code or name for the color of the historic data points.
texcex	numeric variable indicating character size (cex for all text elements in the plot).
caption_above	Boolean variable indicating whether the caption should be drawn above (TRUE) or inside the figure.
family	character string specifying the font family ('serif', 'sans' or 'mono').
no_scenario_numbers	Boolean variable indicating whether climate scenarios should be numbered in the plot (this can clutter the figure).

Value

List of legends for the different panels of the plot. This list reads 'time series labels' for time series plot, 'no adequate labels provided' for unlabeled collections of boxplot, and a data.frame explaining the number codes used as the legend in labeled collections of boxplots. As a side effect, a plot of the climate scenarios is drawn.

The function generates errors, when problems arise.

Author(s)

Eike Luedeling

Examples

```
#making 3 identical objects as scenarios; let's assume these represent the
#years 2000, 2005 and 2010.

models<-list(Chilling_Hours=Chilling_Hours,Utah_Chill_Units=Utah_Model,Chill_Portions=
   Dynamic_Model,GDH=GDH)

chill<-tempResponse(stack_hourly_temps(
   fix_weather(KA_weather[which(KA_weather$Year>2003),]),latitude=50.4),
   Start_JDay = 305,End_JDay = 60,models)
scenario_results<-list(chill,chill,chill)

climate_scenario_list<-list(list(data=scenario_results,
                                 caption=c("Historic","data"),
                                 time_series=TRUE,
                                 labels=c(2000,2005,2010),
                                 historic_data=chill),
                            list(data=scenario_results,
                                 caption=c("Scenario","1"),
                                 labels=c("Climate model 1",
                                          "Climate model 2",
                                          "Climate model 3")),
                            list(data=scenario_results,
                                 caption=c("Scenario","2")),
                            list(data=scenario_results,
                                 caption=c("Scenario","3")))


plot_climate_scenarios(climate_scenario_list,metric="Chill_Portions",
                               metric_label="Chill Portions",
                               year_name="End_year",label_sides="both",
                               reference_line=c(40,45,50),col_rect=NA,col_line=NA,
                               texcex=2,caption_above=FALSE)

plot_climate_scenarios(climate_scenario_list,"Chill_Portions","Chill Portions",
   texcex=1)
```

plot_phenology_trends *Visualizing phenology responses to temperatures during two phases*

Description

The timing of many development stages of temperate trees may depend on temperatures during two phases (e.g. bloom dates depend on the temperature during both the chilling and forcing phase of dormancy). `plot_phenology_trends()` illustrates this dependency as a colored surface with contour lines by applying an interpolating procedure with functions in the `fields` package. The plot is implemented through functions in the `ggplot2` package.

Usage

```
plot_phenology_trends(
  pheno_data,
  weather_data,
  split_month = 6,
  chilling_phase,
  forcing_phase,
  Krig_warn = TRUE,
  x_axis_name = NULL,
  y_axis_name = NULL,
  legend_name = NULL,
  contour_line_color = "black",
  point_color = "black",
  point_shape = 19,
  legend_colors = NULL,
  base_size = 11,
  ...
)
```

Arguments

`pheno_data`	is a data frame that contains information on the timing of phenology events by year. It should consist of two columns called `Year` and `pheno`. Data in the `pheno` column should be in Julian date (day of the year).
`weather_data`	is a data frame containing daily minimum and maximum temperature data (in columns called `Tmin` and `Tmax`, respectively). There also has to be a column for `Year`, one for `Month` and one for `Day`. It can also contain a column for `JDay` (the Julian date, or day of the year).
`split_month`	is an integer representing the last month of the growing season. This procedure analyzes data by phenology year, which can start and end in any month during the calendar year (currently only at the beginning of a month). This variable indicates the last month (e.g. 5 for May) that should be included in the record for a given phenology year. All subsequent months are assigned to the following phenology year.

`chilling_phase` is a vector of integers representing the start and end for the chilling period in temperate trees. Numbers must be provided in Julian date (day of the year).

`forcing_phase` is a vector of integers representing the start and end for the forcing period in temperate trees. Numbers must be provided in Julian date (day of the year).

`Krig_warn` is a boolean parameter passed to the `Krig` function. Default is set to `TRUE` following the recommendation of the authors of the package. For detailed information, please see the documentation of the function.

`x_axis_name` is a character string that allows the user modifying the default label used in the x axis.

`y_axis_name` is a character string that allows the user modifying the default label used in the y axis.

`legend_name` is a character string that allows the user modifying the default label used in the legend.

`contour_line_color` is a character string representing the color used to draw the contour lines. Default is set to black. If `NA` is used, the function will remove the contour lines with a warning.

`point_color` is a character string representing the color used to draw the points for actual observations. Default is set to black. If `NA` is used, the function will remove the points with a warning.

`point_shape` is a numeric input representing the point shape used to draw the points for actual observations. Default is set to 19 (filled point). If `NA` is used, the function will remove the points with a warning.

`legend_colors` is a character string representing the color scale used in the surface plot. Default is set to `NULL` to let the function use the rainbow colors.

`base_size` is a numeric input representing the relative size of the elements in plot. `base_size` is passed to `ggplot2::theme_bw` as well as used to determine the size of the points and contour lines.

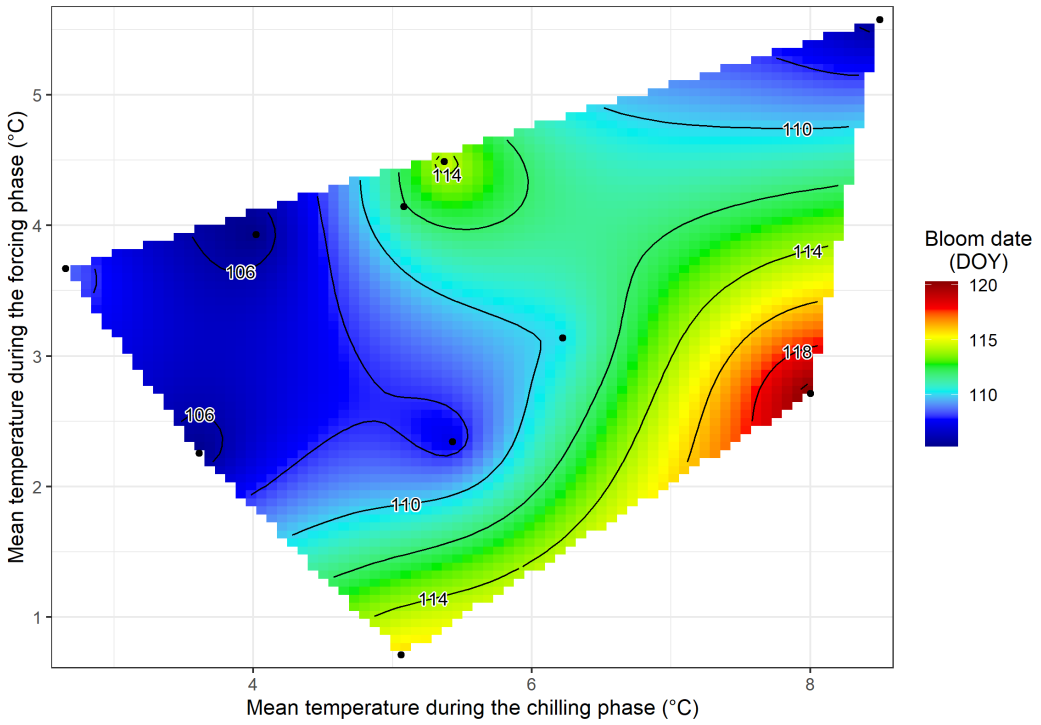
`...` accepts arguments passed to `ggplot2::theme`

Details

The generation of the color surface is based on the Kriging technique, which is typically used for interpolation of spatial data. The use for this particular purpose is a bit experimental.

Value

`plot_phenology_trends()` is expected to return an object of class `gg` and `ggplot`. This means that the plot can be later modified by using the syntax `'+'` from the `ggplot2` package (see examples). The plot returned in the function should look as the following:



Examples

```
# Run a simple plot
# Code is commented out, so that it passes the CRAN incoming checks.
# Please uncomment to run the code.

# plot_phenology_trends(pheno_data = chillR::KA_bloom,
#                       weather_data = chillR::KA_weather,
#                       chilling_phase = c(306, 350),
#                       forcing_phase = c(355, 60))

# Customize the aspects of the plot and save it as 'plot'

# plot <- plot_phenology_trends(pheno_data = chillR::KA_bloom,
#                               weather_data = chillR::KA_weather,
#                               chilling_phase = c(306, 350),
#                               forcing_phase = c(355, 60),
#                            x_axis_name = "Temperatura en el periodo de frio (Celsius)",
#                           y_axis_name = "Temperatura en el periodo de forzado (Celsius)",
#                               legend_name = "Fecha de floracion\n(dia juliano)",
#                               contour_line_color = "white",
#                               point_color = "blue4",
#                               point_shape = 4,
#                               legend_colors = NULL,
#                               base_size = 14,
#                               legend.position = "bottom",
#                               axis.title = ggplot2::element_text(family = "serif"))
```

```
# plot

# Modify the plot object with the syntax from ggplot2.
# Be aware that the following code overrides the modifications
# done by the argument '...' in the main function

# plot + ggplot2::theme_classic(base_size = 14)
```

---

plot_PLS *Output of Partial Least Squares analysis results of phenology vs. daily mean temperatures*

Description

This function produces figures that illustrate statistical correlations between temperature variation during certain phases and the timing of phenological event, based on a PLS analysis conducted with the PLS_pheno or the PLS_chill_force function.

Usage

```
plot_PLS(
  PLS_output,
  PLS_results_path,
  VIP_threshold = 0.8,
  colorscheme = "color",
  plot_bloom = TRUE,
  fonttype = "serif",
  add_chill = c(NA, NA),
  add_heat = c(NA, NA),
  plot_titles_Temp = "Mean temperature",
  plot_titles_chill_force = c("Chill Accumulation", "Heat Accumulation"),
  axis_labels_Temp = expression("Mean temperature ("^"o" * "C)"),
  axis_labels_chill_force = c("Chill Portions per day", "GDH per day"),
  chill_force_same_scale = TRUE
)
```

Arguments

PLS_output	a PLS_output object - the output of the PLS_pheno function. This object is a list with a list element called PLS_summary (and an optional object called PLS_output). This element is a data.frame with the following columns: Date, JDay, Coef, VIP, Tmean, Tstdev. Date is the day of the year in MDD format. JDay is the Julian day (day of the year) of the year in which the biological event is observed; since the analysis will often start in the year before the event, this column often starts with negative numbers. Coef is the coefficient of the PLS regression output. VIP is the Variable Importance in the Projection, another

output of the PLS regression. Tmean is the mean observed temperature of the respective day of the year, for the duration of the phenology record. Tstdev is the standard deviation of temperature on a given day of the year over the length of the phenology record.

`PLS_results_path`
: the path where analysis outputs should be saved. Should include the file name, but without suffix.

`VIP_threshold`
: the VIP threshold, above which a variable is considered important. Defaults to 0.8.

`colorscheme`
: color scheme used for plotting. For grayscale image, this should be set to "bw". Otherwise a color plot is produced.

`plot_bloom`
: boolean variable specifying whether the range of bloom dates should be shown in the plots. If set to TRUE, this range is shown by a semi-transparent gray rectangle. The median bloom date is shown as a dashed line. This only works if the full range of bloom dates is visible in the plot, and it should be set to FALSE if anything other than Julian dates are used as dependent variables.

`fonttype`
: font style to be used for the figure. Can be 'serif' (default) or 'sans'.

`add_chill`
: option for indicating the chilling period in the plot. This should be a numeric vector: c(start_chill,end_chill).

`add_heat`
: option for indicating the forcing period in the plot. This should be a numeric vector: c(start_heat,end_heat).

`plot_titles_Temp`
: title for the bottom plot, which relates PLS outputs to values of the input variable (temperature in the original version). Only affects the output for PLS_Temp_pheno objects.

`plot_titles_chill_force`
: titles for the bottom plots, which relate PLS outputs to values of the input variables (chill and heat accumulation). Only affects the output for PLS_chillforce_pheno objects.

`axis_labels_Temp`
: y-axis label for the bottom plot, which relates PLS outputs to values of the input variable (temperature in the original version). Only affects the output for PLS_Temp_pheno objects.

`axis_labels_chill_force`
: y-axis labels for the bottom plots, which relate PLS outputs to values of the input variables (chill and heat accumulation). Only affects the output for PLS_chillforce_pheno objects.

`chill_force_same_scale`
: Boolean parameter indicating whether the two sets of VIP scores and model coefficients resulting from a PLS_chillforce_pheno analysis should be shown on the same scale in the separate output diagrams. Since this is generally advisable for comparison, this defaults to TRUE.

Details

Ths figure illustrates results from the PLS_pheno function, which uses Partial Least Squares (or Projection to Latent Structures) regression to examine the relationship between mean daily temper-

atures and the timing of an annual biological event. It produces a plot (as a bmp image) with three panels: the top panel shows the value of the VIP score for each day of the year; the middle panel shows the model coefficients and the bottom panel shows the mean temperature and its standard deviation. In the top plot, all days with VIP scores above VIP_threshold are shown in blue. In the other two panels, values for the same days are shown in red, which high VIP scores coincide with negative model coefficients, and in green for positive coefficients. This function does not produce an output, but as side effects it produces a bmp image and a table that summarizes all data used for making the figure in the specified folder.

Author(s)

Eike Luedeling

References

The method is described here:

Luedeling E and Gassner A, 2012. Partial Least Squares Regression for analyzing walnut phenology in California. Agricultural and Forest Meteorology 158, 43-52.

Wold S, 1995. PLS for multivariate linear modeling. In: van der Waterbeemd H (ed) Chemometric methods in molecular design: methods and principles in medicinal chemistry, vol 2. Chemie, Weinheim, pp 195-218.

Wold S, Sjostrom M, Eriksson L, 2001. PLS-regression: a basic tool of chemometrics. Chemometr Intell Lab 58(2), 109-130.

Mevik B-H, Wehrens R, Liland KH, 2011. PLS: Partial Least Squares and Principal Component Regression. R package version 2.3-0. http://CRAN.R-project.org/package0pls.

Some applications:

Guo L, Dai J, Wang M, Xu J, Luedeling E, 2015. Responses of spring phenology in temperate zone trees to climate warming: a case study of apricot flowering in China. Agricultural and Forest Meteorology 201, 1-7.

Luedeling E, Kunz A and Blanke M, 2013. Identification of chilling and heat requirements of cherry trees - a statistical approach. International Journal of Biometeorology 57,679-689.

Yu H, Luedeling E and Xu J, 2010. Stronger winter than spring warming delays spring phenology on the Tibetan Plateau. Proceedings of the National Academy of Sciences (PNAS) 107 (51), 22151-22156.

Yu H, Xu J, Okuto E and Luedeling E, 2012. Seasonal Response of Grasslands to Climate Change on the Tibetan Plateau. PLoS ONE 7(11), e49230.

Examples

```
weather<-fix_weather(KA_weather[which(KA_weather$Year>2004),])
#Plots look much better with weather<-fix_weather(KA_weather)
#but that takes to long to run for passing CRAN checks

PLS_results<-PLS_pheno(
  weather_data=weather$weather,
  split_month=6,   #last month in same year
  bio_data=KA_bloom)
```

```
PLS_results_path<-paste(getwd(),"/PLS_output",sep="")

#plot_PLS(PLS_results,PLS_results_path)
#plot_PLS(PLS_results,PLS_results_path,add_chill=c(307,19),add_heat=c(54,109))

dc<-daily_chill(stack_hourly_temps(weather,50.4), 11)
plscf<-PLS_chill_force(daily_chill_obj=dc, bio_data_frame=KA_bloom, split_month=6)

#plot_PLS(plscf,PLS_results_path)
#plot_PLS(plscf,PLS_results_path,add_chill=c(307,19),add_heat=c(54,109))
```

`plot_scenarios` *Plot historic and future scenarios for climate-related metrics (ggplot2 version)*

Description

Visualize outputs from the `temperature_generation` function used in climate-related assessments. These outputs are usually compiled with the `make_climate_scenario` function.

Usage

```
plot_scenarios(
  scenario_list,
  metric,
  add_historic = TRUE,
  ...,
  outlier_shape = 19,
  historic_color = "white",
  group_by = c("Scenario", "Year"),
  y_axis_name = paste("Cumulative response in", metric),
  x_axis_name = "Year",
  legend_title = "Climate model",
  legend_labels = NULL,
  panel_labels = NULL,
  base_size = 11
)
```

Arguments

`scenario_list`	is a list of lists containing information and data about the scenarios to be plotted. These lists must have:

- an element named `data`, which should be a list containing one or more named dataframes with a column named the same as the `metric` argument. This column must contain (`numeric`) information to be plotted. Dataframes of climate-related metrics can be obtained with the `tempResponse_daily_list` function. For past scenarios, the names of the dataframes can be the reference years used to generate the scenarios. These names will be recycled and used in the x-axis of the historic panel. For future scenarios, the names of the dataframes can be the models used in the projections. These names will appear in the legend for future panels.
- an element named `caption` containing information about the scenario which the list is related to.
- an element named `historic_data` which represents a data frame for actual observations in past scenarios. This element can be optional but is mandatory if `add_historic = TRUE`
- `time_series` is an optional argument that defines whether the scenario contains a time series.
- `labels` is an optional vector that usually contains the names of the elements used for `metric_summary` in `make_climate_scenario`.

`metric` is a character string corresponding to the name of the column that contains the data of interest in the dataframe of the `scenario_list` (and, if applicable, in the `historic_data`).

`add_historic` is a boolean parameter to define whether the plot should include the actual observations of historic climate-related metrics.

`...` accepts arguments that can be passed to `layer` and are commonly used outside the aesthetic function for different geoms. In this case, `...` is passed to the `geom_point` function in the case that actual observations of chill or heat are displayed. Options are `size`, `color`, among others.

`outlier_shape` is the optional shape to replace the outliers in the boxplots. To show no oultiers use `NA`. See `shape` for shape options.

`historic_color` is a character string corresponding to the color used to fill the boxplots in simulated historic scenarios. Supported options are those provided by `colors`.

`group_by` is a vector of character strings indicating how the plots should be grouped. I.e. by `Scenario` and then `Year` or viceversa.

`y_axis_name` is a character string representing the title of the y axis in the final plot. Default is set to `paste('Cumulative response in', metric)` to let the function obtain the name based on the `metric` argument.

`x_axis_name` is a character string representing the title of the x axis in the 'Historic' panel. Default is set to `Year`.

`legend_title` is a character string representing the title of the legend showing the climate models used in the assessment.

`legend_labels` is a vector of character strings that allows the user to modify the names of the climate models used in the projections. The length of the vector must coincide with the number of climate models. Default is set to `NULL` to let the function use the labels generated with the `make_climate_scenario` function.

`panel_labels` is a list of 3 named objects that allows the user to customize the text in the upper part of the plot. Default is set to `NULL` to let the function use the labels generated with the `make_climate_scenario` function. If provided, the objects of the list must be:

- an element named **Historic** containing the name to be used in the 'Historic' panel.
- an element named **Scenario** containing the names of the scenarios used for the projections. If `group_by = c("Year", "Scenario")` is used, `Scenario` must be a list of named objects according to the labels used in the `Year` object. See examples.
- an element named **Year** containing the labels to be used for the time horizons used in the assessment. If `group_by = c("Scenario", "Year")` is used, `Year` must be a list of named objects according to the labels used in the `Scenario` object. See examples.

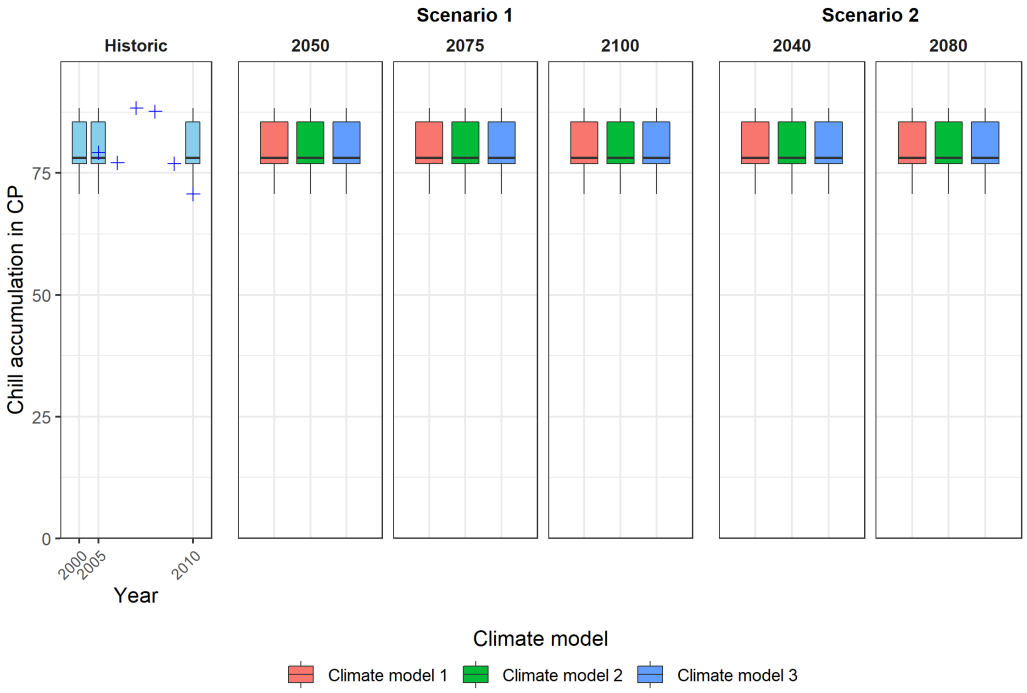
`base_size` is an integer to define the relative size of the text in the final plot. This argument is passed to `ggpplot2::theme_bw`. Default is set to 11.

Details

`plot_scenarios` uses the `ggplot2` syntax for producing separated plots for historic and future scenarios. Later, the plots are merged into one final figure by using the `patchwork` library.

Value

A plot of classes `'patchwork'`, `'gg'`, and `'ggplot'`. This allows the user to continue editing some features of the plots through the syntax (i.e. `'&'`, and `'+'`) from both libraries (see examples).



Author(s)

Eduardo Fernandez and Eike Luedeling

Examples

```
# Make 3 identical objects as scenarios; let's assume these represent the
# years 2000, 2005 and 2010.

library(chillR)

# Compute chill responses for KA_weather data

chill <- tempResponse(stack_hourly_temps(
                    fix_weather(KA_weather[which(KA_weather$Year > 2006), ]),
                    latitude = 50.4), Start_JDay = 305, End_JDay = 60)


# Simulated scenarios labels

past_labels <- c(2000, 2005, 2010)

# Models labels

models_labels <- c("Climate model 1", "Climate model 2",
                   "Climate model 3")
```

```
# Add named elements to past and future scenarios

scenario_results_past <- list(`2000` = chill,
                              `2005` = chill,
                              `2010` = chill)

scenario_results_future <- list(`Climate model 1` = chill,
                                `Climate model 2` = chill,
                                `Climate model 3` = chill)

# Define the climate scenario

climate_scenario_list <- list(list(data = scenario_results_past,
                                   caption = c("Historic", "data"),
                                   time_series = TRUE,
                                   labels = past_labels,
                                   historic_data = chill),
                              list(data = scenario_results_future,
                                   caption = c("Scenario 1", "2050"),
                                   labels = models_labels),
                              list(data = scenario_results_future,
                                   caption = c("Scenario 1", "2075"),
                                   labels = models_labels),
                              list(data = scenario_results_future,
                                   caption=c("Scenario 1", "2100"),
                                   labels = models_labels),
                              list(data = scenario_results_future,
                                   caption=c("Scenario 2", "2040"),
                                   labels = models_labels),
                              list(data = scenario_results_future,
                                   caption=c("Scenario 2", "2080"),
                                   labels = models_labels))

# Plot the climate scenarios

plot_scenarios(climate_scenario_list, metric = 'Chill_Portions',
               add_historic = TRUE, size = 2, shape = 3, color = 'blue',
               outlier_shape = 12, historic_color = 'skyblue',
               group_by = c("Year", "Scenario"))

## Plot scenarios modifying the whole text in the plot
## We will comment the next examples to reduce the running time in CRAN
## submissions...
# plot_scenarios(scenario_list = climate_scenario_list, metric = 'Chill_Portions',
#                add_historic = TRUE, size = 2, shape = 3, color = 'blue',
#                outlier_shape = 12, historic_color = 'skyblue',
#                group_by = c("Scenario", "Year"),
#                y_axis_name = "Acumulacion de frio en CP",
#                x_axis_name = "Tiempo",
#                legend_title = "Modelo climatico",
#                legend_labels = c("Modelo 1", "Modelo 2", "Modelo 3"),
#                panel_labels = list(Historic = "Historico",
#                                    Scenario = c("Escenario 1",
```

```
#                                              "Escenario 2"),
#                                   Year = list(`Escenario 1` = c("Futuro cercano",
#                                                                 "Futuro medio",
#                                                                 "Future lejano"),
#                                               `Escenario 2` = c("Futuro cercano",
#                                                                 "Futuro medio"))))

## Since the output is a ggplot object, it is possible to continue
## modifying some general aspects of the plot

## Define the basic plot
# plot <- plot_scenarios(climate_scenario_list, metric = 'Chill_Portions',
#                        add_historic = TRUE, size = 2, shape = 3, color = 'blue',
#                        outlier_shape = 12, historic_color = 'skyblue')


## Example to change the color of the climate model scale

# plot & ggplot2::scale_fill_brewer(type = 'qual')

## Modify the format of axis title and axis text

# plot & ggplot2::theme(axis.title = ggplot2::element_text(size = 14,
#                                                          family = 'serif'),
#                       axis.text = ggplot2::element_text(face = 'bold',
#                                                         color = 'blue'))
```

PLS_chill_force *Partial Least Squares analysis of phenology vs. accumulated daily chill and heat*

Description

This function conducts a Partial Least Squares (PLS) regression analysis relating an annual biological phenomenon, e.g. fruit tree flowering or leaf emergence, to mean daily rates of chill (with three models) and heat accumulation of the preceding 12 months. It produces figures that illustrate statistical correlations between temperature variation during certain phases and the timing of phenological events.

Usage

```
PLS_chill_force(
  daily_chill_obj,
  bio_data_frame,
  split_month,
  expl.var = 30,
  ncomp.fix = NULL,
  return.all = FALSE,
```

```
  crossvalidate = "none",
  end_at_pheno_end = TRUE,
  chill_models = c("Chilling_Hours", "Utah_Chill_Units", "Chill_Portions"),
  heat_models = c("GDH"),
  runn_means = 1,
  metric_categories = c("Chill", "Heat")
)
```

Arguments

daily_chill_obj	a daily chill object. This should be generated with the daily_chill function.
bio_data_frame	a data frame that contains information on the timing of phenological events by year. It should consist of two columns called Year and pheno. Data in the pheno column should be in Julian date (day of the year).
split_month	the procedure analyzes data by phenological year, which can start and end in any month during the calendar year (currently only at the beginning of a month). This variable indicates the last month (e.g. 5 for May) that should be included in the record for a given phenological year. All subsequent months are assigned to the following phenological year.
expl.var	percentage of the variation in the dependent variable that the PLS model should explain. This is used as a threshold in finding the appropriate number of components in the PLS regression procedure.
ncomp.fix	fixed number of components for the PLS model. Defaults to NULL, so that the number is automatically determined, but it can also be set by the user.
return.all	boolean variable indicating whether or not the full set of PLS results should be returned by the function. If this is set to TRUE, the function output is a list with two elements (besides the object_type string): PLS_summary and PLS_output; if it is set to FALSE, only the PLS_summary is returned.
crossvalidate	character variable indicating what kind of validation should be performed by the PLS procedure. This defaults to "none", but the plsr function (of the pls package) also takes "CV" and "LOO" as inputs. See the documentation for the plsr function for details.
end_at_pheno_end	boolean variable indicating whether the analysis should disregard temperatures after the last date included in the bio_data_frame dataset. If set to TRUE, only temperatures up this date are considered. Phenology data is extracted from the PLS output files. If this parameter is assigned a numeric value, only data up to the Julian date specified by this number are considered.
chill_models	Character vector containing names of chill models that should be considered in the PLS regression. These names should correspond to column names of daily_chill. This defaults to c("Chilling_Hours", "Utah_Chill_Units", "Chill_Portions").
heat_models	Character vector containing names of heat models that should be considered in the PLS regression. These names should correspond to column names of daily_chill. This defaults to c("GDH").

runn_means numeric vector specifying whether inputs to the PLS calculation should be processed by a running mean filter. This usually enhances the clarity of results. This vector contains either one element (an integer), in which case the same filter is applied to all input metrics, or one element for each model, which allows specifying metric-specific running means. In this case the sequence of numbers should correspond to the sequence specified in the function call, with chill models listed first, followed by heat models.

metric_categories
while the original application of this function is the calculation of tree responses to chill and heat accumulation, it can also be applied for other variables. In this case, you may not want the outputs to be called 'Chill' and 'Heat' (the default). Here you can specify a character vector of length 2, which contains the labels you want to appear in the output table.

Details

PLS regression is useful for exploring the relationship between daily chill and heat accumulation rates and biological phenomena that only occur once per year. The statistical challenge is that a normally quite small number of observations must be related to variation in a much larger number (730) of daily chill and heat values, which are also highly autocorrelated. Most regression approaches are not suitable for this, but PLS regression offers a potential solution. The method is frequently used in chemometrics and hyperspectral remote sensing, where similar statistical challenges are encountered. The basic mechanism is that PLS first constructs latent factors (similar to principal components) from the independent data (daily chill and heat accumulation) and then uses these components for the regression. The contribution of each individual variable to the PLS model is then evaluated with two main metrics: the Variable Importance in the Projection statistic (VIP) indicates how much variation in a given independent variable is correlated with variation in the dependent variable. A threshold of 0.8 is often used for determining importance. The standardized model coefficients of the PLS model then give an indication of the direction and strength of the effect, e.g. if coefficients are positive and high, high values for the respective independent variable are correlated with high values of the dependent variable (e.g. late occurrence of a phenological stage). This procedure was inspired by the challenge of explaining variation in bloom and leaf emergence dates of temperate fruit trees in Mediterranean climates. These are generally understood to result from (more of less) sequential fulfillment of a chilling and a forcing requirement. During the chilling phase, cool temperatures are needed; during the forcing phase, trees need heat. There is no easily visible change in tree buds that would indicate the transition between these two phases, making it difficult to develop a good model of these processes. Where long-term phenology data are available and can be coupled with daily chill and heat records (derived from daily temperature data), PLS regression allows detection of the chilling/forcing transition. This procedure has not often been applied to biological phenomena at the time of writing this, and there may be constraints to how generally applicable it is. Yet is has passed the test of scientific peer review a few times, and it has produced plausible results in a number of settings. This package draws heavily from the pls package.

Per default, chill metrics used are the ones given in the references below. Chilling Hours are all hours with temperatures between 0 and 7.2 degrees C. Units of the Utah Model are calculated as suggested by Richardson et al. (1974) (different weights for different temperature ranges, and negation of chilling by warm temperatures). Chill Portions are calculated according to Fishman et al. (1987a,b). More honestly, they are calculated according to an Excel sheet produced by Amnon

Erez and colleagues, which converts the complex equations in the Fishman papers into relatively simple Excel functions. These were translated into R. References to papers that include the full functions are given below. Growing Degree Hours are calculated according to Anderson et al. (1986), using the default values they suggest.

It is possible, however, for the user to specify other metrics to be evaluated. These should be indicated by the chill_models and heat_models parameters, which should contain the names of the respective columns of the daily_chill_obj$daily_chill data frame.

Value

`object_type` the character string "PLS_chillforce_pheno". This is only needed for choosing the correct method for the plot_PLS function.

`pheno` a data frame containing the phenology data used for the PLS regression, with columns Year and pheno.

`<chill_model>$<heat_model>` for each combination of elements from chill_models and heat_models, a list element is generated, which contains a list with elements PLS_summary and (if(return.all=TRUE) PLS_output. These contain the results of the PLS analysis that used the respective chill and heat metrics as independent variables.

Note

After doing extensive model comparisons, and reviewing a lot of relevant literature, I do not recommend using the Chilling Hours or Utah Models, especially in warm climates! The Dynamic Model (Chill Portions), though far from perfect, seems much more reliable.

Author(s)

Eike Luedeling, with contributions from Sabine Guesewell

References

Model references, for the default option:

Chilling Hours:

Weinberger JH (1950) Chilling requirements of peach varieties. Proc Am Soc Hortic Sci 56, 122-128

Bennett JP (1949) Temperature and bud rest period. Calif Agric 3 (11), 9+12

Utah Model:

Richardson EA, Seeley SD, Walker DR (1974) A model for estimating the completion of rest for Redhaven and Elberta peach trees. HortScience 9(4), 331-332

Dynamic Model:

Erez A, Fishman S, Linsley-Noakes GC, Allan P (1990) The dynamic model for rest completion in peach buds. Acta Hortic 276, 165-174

Fishman S, Erez A, Couvillon GA (1987a) The temperature dependence of dormancy breaking in plants - computer simulation of processes studied under controlled temperatures. J Theor Biol 126(3), 309-321

Fishman S, Erez A, Couvillon GA (1987b) The temperature dependence of dormancy breaking in plants - mathematical analysis of a two-step model involving a cooperative transition. J Theor Biol 124(4), 473-483

Growing Degree Hours:

Anderson JL, Richardson EA, Kesner CD (1986) Validation of chill unit and flower bud phenology models for 'Montmorency' sour cherry. Acta Hortic 184, 71-78

Model comparisons and model equations:

Luedeling E, Zhang M, Luedeling V and Girvetz EH, 2009. Sensitivity of winter chill models for fruit and nut trees to climatic changes expected in California's Central Valley. Agriculture, Ecosystems and Environment 133, 23-31

Luedeling E, Zhang M, McGranahan G and Leslie C, 2009. Validation of winter chill models using historic records of walnut phenology. Agricultural and Forest Meteorology 149, 1854-1864

Luedeling E and Brown PH, 2011. A global analysis of the comparability of winter chill models for fruit and nut trees. International Journal of Biometeorology 55, 411-421

Luedeling E, Kunz A and Blanke M, 2011. Mehr Chilling fuer Obstbaeume in waermeren Wintern? (More winter chill for fruit trees in warmer winters?). Erwerbs-Obstbau 53, 145-155

Review on chilling models in a climate change context:

Luedeling E, 2012. Climate change impacts on winter chill for temperate fruit and nut production: a review. Scientia Horticulturae 144, 218-229

The PLS method is described here:

Luedeling E and Gassner A, 2012. Partial Least Squares Regression for analyzing walnut phenology in California. Agricultural and Forest Meteorology 158, 43-52.

Wold S (1995) PLS for multivariate linear modeling. In: van der Waterbeemd H (ed) Chemometric methods in molecular design: methods and principles in medicinal chemistry, vol 2. Chemie, Weinheim, pp 195-218.

Wold S, Sjostrom M, Eriksson L (2001) PLS-regression: a basic tool of chemometrics. Chemometr Intell Lab 58(2), 109-130.

Mevik B-H, Wehrens R, Liland KH (2011) PLS: Partial Least Squares and Principal Component Regression. R package version 2.3-0. http://CRAN.R-project.org/package0pls.

Some applications of the PLS procedure:

Luedeling E, Kunz A and Blanke M, 2013. Identification of chilling and heat requirements of cherry trees - a statistical approach. International Journal of Biometeorology 57,679-689.

Yu H, Luedeling E and Xu J, 2010. Stronger winter than spring warming delays spring phenology on the Tibetan Plateau. Proceedings of the National Academy of Sciences (PNAS) 107 (51), 22151-22156.

Yu H, Xu J, Okuto E and Luedeling E, 2012. Seasonal Response of Grasslands to Climate Change on the Tibetan Plateau. PLoS ONE 7(11), e49230.

The exact procedure was used here:

Luedeling E, Guo L, Dai J, Leslie C, Blanke M, 2013. Differential responses of trees to temperature variation during the chilling and forcing phases. Agricultural and Forest Meteorology 181, 33-42.

The chillR package:

Luedeling E, Kunz A and Blanke M, 2013. Identification of chilling and heat requirements of cherry trees - a statistical approach. International Journal of Biometeorology 57,679-689.

Examples

```
weather<-fix_weather(KA_weather[which(KA_weather$Year>2004),])
#Plots look much better with weather<-fix_weather(KA_weather)
#but that takes too long to run for passing CRAN checks

dc<-daily_chill(stack_hourly_temps(weather,50.4), 11)
plscf<-PLS_chill_force(daily_chill_obj=dc, bio_data_frame=KA_bloom, split_month=6)

#PLS_results_path<-paste(getwd(),"/PLS_output",sep="")
#plot_PLS(plscf,PLS_results_path)
#plot_PLS(plscf,PLS_results_path,add_chill=c(307,19),add_heat=c(54,109))
```

PLS_pheno *Partial Least Squares analysis of phenology vs. daily mean temperatures*

Description

This function conducts a Partial Least Squares (PLS) regression analysis relating an annual biological phenomenon, e.g. fruit tree flowering or leaf emergence, to mean daily temperatures of the preceding 12 months. It produces figures that illustrate statistical correlations between temperature variation during certain phases and the timing of phenological event.

Usage

```
PLS_pheno(
  weather_data,
  bio_data,
  split_month = 7,
  runn_mean = 11,
  expl.var = 30,
  ncomp.fix = NULL,
  use_Tmean = FALSE,
  return.all = FALSE,
  crossvalidate = "none",
  end_at_pheno_end = TRUE
)
```

Arguments

weather_data	a dataframe containing daily minimum and maximum temperature data (in columns called Tmin and Tmax, respectively), and/or mean daily temperature (in a column called Tmean). There also has to be a column for Year and one for JDay (the Julian date, or day of the year). Alternatively, the date can also be given in three columns (Years, Month and Day).

`bio_data` a data frame that contains information on the timing of phenological events by year. It should consist of two columns called Year and pheno. Data in the pheno column should be in Julian date (day of the year).

`split_month` the procedure analyzes data by phenological year, which can start and end in any month during the calendar year (currently only at the beginning of a month). This variable indicates the last month (e.g. 5 for May) that should be included in the record for a given phenological year. All subsequent months are assigned to the following phenological year.

`runn_mean` application of a running mean function to daily mean temperatures before running the PLS procedure substantially enhances the clarity of outputs. runn_mean requires an odd integer value specifying how many days should be included in this running mean. runn_mean=11 has usually produced good results.

`expl.var` percentage of the variation in the dependent variable that the PLS model should explain. This is used as a threshold in finding the appropriate number of components in the PLS regression procedure.

`ncomp.fix` fixed number of components for the PLS model. Defaults to NULL, so that the number is automatically determined, but is can also be set by the user.

`use_Tmean` boolean variable indicating whether or not the column Tmean from the weather_data_frame should be used as input for the PLS analysis. If this is set to FALSE, Tmean is calculated as the arithmetic mean of Tmin and Tmax.

`return.all` boolean variable indicating whether or not the full set of PLS results should be output from the function. If this is set to TRUE, the function output is a list with two elements: PLS_summary and PLS_output; if it is set to FALSE, only the PLS_summary is returned.

`crossvalidate` character variable indicating what kind of validation should be performed by the PLS procedure. This defaults to "none", but the plsr function (of the pls package) also takes "CV" and "LOO" as inputs. See the documentation for the plsr function for details.

`end_at_pheno_end`

boolean variable indicating whether the analysis should disregard temperatures after the last date included in the bio_data_frame dataset. If set to TRUE, only temperatures up this date are considered. Phenology data is extracted from the PLS output files. If this parameter is assigned a numeric value, only data up to the Julian date specified by this number are considered.

Details

PLS regression is useful for exploring the relationship between daily temperature data and biological phenomena that only occur once per year. The statistical challenge is that a normally quite small number of observations must be related to variation in a much larger number (365) of daily temperatures, which are also highly autocorrelated. Most regression approaches are not suitable for this, but PLS regression offers a potential solution. The method is frequently used in chemometrics and hyperspectral remote sensing, where similar statistical challenges are encountered. The basic mechanism is that PLS first constructs latent factors (similar to principal components) from the independent data (temperatures) and then uses these components for the regression. The contribution of each individual variable to the PLS model is then evaluated with two main metrics: the Variable

Importance in the Projection statistic (VIP) indicates how much variation in a given independent variable is correlated with variation in the dependent variable. A threshold of 0.8 is often used for determining importance. The standardized model coefficients of the PLS model then give an indication of the direction and strength of the effect, e.g. if coefficients are positive and high, high values for the respective independent variable are correlated with high values of the dependent variable (e.g. late occurrence of a phenological stage). This procedure was inspired by the challenge of explaining variation in bloom and leaf emergence dates of temperate fruit trees in Mediterranean climates. These are generally understood to result from (more of less) sequential fulfillment of a chilling and a forcing requirement. During the chilling phase, cool temperatures are needed; during the forcing phase, trees need heat. There is no easily visible change in tree buds that would indicate the transition between these two phases, making it difficult to develop a good model of these processes. Where long-term phenology data are available and can be couple with daily temperature records, PLS regression allows detection of the chilling/forcing transition. This procedure has not often been applied to biological phenomena at the time of writing this, and there may be constraints to how generally applicable it is. Yet is has passed the test of scientific peer review a few times, and it has produced plausible results in a number of settings. This package draws heavily from the pls package. It also incorporates very helpful comments from Sabine Guesewell of ETH Zurich (Switzerland), who pointed out some errors in the PLS procedure and made suggestions for improvement.

Value

`object_type`	the character string "PLS_Temp_pheno". This is only needed for choosing the correct method for the plot_PLS function.
`pheno`	a data frame containing the phenology data used for the PLS regression, with columns Year and pheno.
`PLS_summary`	a data frame containing all important outputs of the PLS regression. Columns are Date (in MMDD format), JDay (Julian date, or day of the year), Coefficient (the PLS model coefficient for each daily temperature variable), and VIP (the Variable Importance in the Projection score). The columns Tmean and Tstdev contain the means and standard deviations of temperature for each day of the year.
`PLS_output`	this is the complete output of the plsr function of the pls package. See the documentation for that package for further details.

Author(s)

Eike Luedeling, with contributions from Sabine Guesewell

References

The method is described here:

Luedeling E and Gassner A, 2012. Partial Least Squares Regression for analyzing walnut phenology in California. Agricultural and Forest Meteorology 158, 43-52.

Wold S (1995) PLS for multivariate linear modeling. In: van der Waterbeemd H (ed) Chemometric methods in molecular design: methods and principles in medicinal chemistry, vol 2. Chemie, Weinheim, pp 195-218.

Wold S, Sjostrom M, Eriksson L (2001) PLS-regression: a basic tool of chemometrics. Chemometr Intell Lab 58(2), 109-130.

Mevik B-H, Wehrens R, Liland KH (2011) PLS: Partial Least Squares and Principal Component Regression. R package version 2.3-0. http://CRAN.R-project.org/package0pls.

Some applications:

Luedeling E, Kunz A and Blanke M, 2013. Identification of chilling and heat requirements of cherry trees - a statistical approach. International Journal of Biometeorology 57,679-689.

Yu H, Luedeling E and Xu J, 2010. Stronger winter than spring warming delays spring phenology on the Tibetan Plateau. Proceedings of the National Academy of Sciences (PNAS) 107 (51), 22151-22156.

Yu H, Xu J, Okuto E and Luedeling E, 2012. Seasonal Response of Grasslands to Climate Change on the Tibetan Plateau. PLoS ONE 7(11), e49230.

Examples

```
PLS_results<-PLS_pheno(
  weather_data=KA_weather,
  split_month=6,   #last month in same year
  bio_data=KA_bloom)

PLS_results_path<-paste(getwd(),"/PLS_output",sep="")

# plot_PLS(PLS_results,PLS_results_path)
```

---

predict.bootstrap_phenologyFit

predict bootstrap_phenologyFit

Description

Generic function to predict a ‘bootstrap_phenologyFit‘ object.

Usage

```
## S3 method for class 'bootstrap_phenologyFit'
predict(object, SeasonList, ...)
```

Arguments

object	object of class ‘phenologyFit‘ to predict.
SeasonList	List with data frames per season, see phenologyFit for more details.
...	additional parameters, ignored here

Value

A data.frame with one column ‘pbloomJDays‘ and a second one ‘Err‘.

predict.phenologyFit *predict phenologyFit*

Description

Generic function to predict a ‘phenologyFit‘ object.

Usage

```
## S3 method for class 'phenologyFit'
predict(object, SeasonList, ...)
```

Arguments

object	object of class ‘phenologyFit‘ to predict.
SeasonList	List with data frames per season, see phenologyFit for more details.
...	additional parameters, ignored here

Value

A numeric vector is returned with a predicted bloom day per Season in ‘SeasonList‘. If ‘SeasonList‘ is missing, the original ‘SeasonList‘ is used for prediction.

print.phenologyFit *print phenologyFit*

Description

print phenologyFit

Usage

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

Arguments

x	class phenologyFit. object to print
...	additional parameters, ignored here

Value

No return value.

read_tab *Read csv table regardless of whether it is a true csv or the French type*

Description

csv tables are widely used for storing data as 'comma-separated values'. This doesn't work, however, when the comma is also used as a decimal symbol, as is practiced in French or German, for example. The separator symbol for csv files then becomes a semi-colon. This is not problematic when you only work on one machine, but it causes problems when you collaborate with people who use different types of csv encoding.

Usage

```
read_tab(tab)
```

Arguments

tab	file name of a table to be read.

Details

This function overcomes this problem by checking first, which of the two characters occurs most frequently in the table, assuming then that this is the separator symbol. It then opens the table accordingly.

Currently limited to files that are either comma-separated with point as decimal symbol or semicolon-separated with comma as decimal symbol. Files should also have a header.

Value

If the table is in one of the two formats described above, the stored table is returned.

Author(s)

Eike Luedeling

Examples

```
df<-data.frame(Var1=c(1,2,3.2,1.2),Var2=c(1.2,6,2.6,7))
write.csv(df,"filecsv.csv",row.names=FALSE)
read_tab("filecsv.csv")
write.table(df,"filesemicolon.csv",sep=";",dec=",")
read_tab("filesemicolon.csv")
file.remove("filecsv.csv")
file.remove("filesemicolon.csv")
```

RMSEP *Root Mean Square Error of Prediction (RMSEP)*

Description

This function computes the Root Mean Square Error of Prediction (RMSEP), a commonly used measure for the predictive capacity of a model. It compares values predicted with a model with observed values.

Usage

```
RMSEP(predicted, observed, na.rm = FALSE)
```

Arguments

predicted	a numeric vector containing predicted values.
observed	a numeric vector of the same length as "'predicted"' containing observed values.
na.rm	Boolean parameter indicating whether NA values should be removed before the analysis

Value

numeric value of the RMSEP.

Author(s)

Eike Luedeling

Examples

```
predicted<-c(1,2,3,4,5,6,7,8,9,10)
observed<-c(1.5,1.8,3.3,3.9,4.4,6,7.5,9,11,10)

RMSEP(predicted,observed)
```

RPD *Residual Prediction Deviation (RPD)*

Description

This function computes the Residual Prediction Deviation (RPD), which is defined as the standard deviation of observed values divided by the Root Mean Square Error or Prediction (RMSEP). The RDP takes both the prediction error and the variation of observed values into account, providing a metric of model validity that is more objective than the RMSEP and more easily comparable across model validation studies. The greater the RPD, the better the model's predictive capacity.

Usage

```
RPD(predicted, observed, na.rm = FALSE)
```

Arguments

predicted	a numeric vector containing predicted values.
observed	a numeric vector of the same length as "'predicted"' containing observed values.
na.rm	Boolean parameter indicating whether NA values should be removed before the analysis

Details

Interpretation of the RPD is somewhat arbitrary, with different thresholds for a good model used in the literature. Many studies call a model *excellent*, when the RPD is above 2 (but other classification use thresholds as high as 8 for this).

Value

numeric value of the RDP.

Author(s)

Eike Luedeling

References

Williams PC and Sobering DC (1993) Comparison of commercial near infrared transmittance and reflectance instruments for analysis of whole grains and seeds. J. Near Infrared Spectrosc. 1, 25-32 (I didn't have access to this paper, but have noticed that it is often provided as the key reference for the RPD).

Examples

```
predicted<-c(1,2,3,4,5,6,7,8,9,10)
observed<-c(1.5,1.8,3.3,3.9,4.4,6,7.5,9,11,10)

RPD(predicted,observed)
```

RPIQ *Ratio of Performance to InterQuartile distance (RPIQ)*

Description

This function computes the Ratio of Performance to InterQuartile distance (RPIQ), which is defined as interquartile range of the observed values divided by the Root Mean Square Error or Prediction (RMSEP). The RPIQ takes both the prediction error and the variation of observed values into account, providing a metric of model validity that is more objective than the RMSEP and more easily comparable across model validation studies. The greater the RPIQ, the better the model's predictive capacity. In contrast to the Residual Prediction Deviation (RPD), the RPIQ makes no assumptions about the distribution of the observed values (since the RDP includes a standard deviation, it assumed normal distribution of the observed values).

Usage

```
RPIQ(predicted, observed, na.rm = FALSE)
```

Arguments

predicted	a numeric vector containing predicted values.
observed	a numeric vector of the same length as "'predicted"' containing observed values.
na.rm	Boolean parameter indicating whether NA values should be removed before the analysis

Details

Interpretation of the RPIQ differs in the literature, with different thresholds used for judging model quality.

Value

numeric value of the RPIQ

Author(s)

Eike Luedeling

References

Bellon-Maurel V, Fernandez-Ahumada E, Palagos B, Roger J-M, McBratney A, 2010. Critical review of chemometric indicators commonly used for assessing the quality of the prediction of soil attributes by NIR spectroscopy, In TrAC Trends in Analytical Chemistry 29(9), 1073-1081.

Examples

```
predicted<-c(1,2,3,4,5,6,7,8,9,10)
observed<-c(1.5,1.8,3.3,3.9,4.4,6,7.5,9,11,10)

RPD(predicted,observed)
```

---

runn_mean *Running mean of a vector*

Description

Function to calculate the running mean of a numeric vector

Usage

```
runn_mean(
  vec,
  runn_mean,
  na.rm = FALSE,
  exclude_central_value = FALSE,
  FUN = mean
)
```

Arguments

vec	numeric vector
runn_mean	number of vector elements to use for calculating the running mean
na.rm	ignore NA values when calculating means. Defaults to FALSE.
exclude_central_value	exclude central value in calculating means. Defaults to FALSE.
FUN	function to be applied. For a running mean, this is usually mean (the default), but other functions can also be specified here (the na.rm parameter won't work then, and the function has to be dependent on one numeric variable only.

Value

numeric vector containing the running mean

Author(s)

Eike Luedeling

Examples

```
plot(runn_mean(rnorm(1000),150))
```

runn_mean_pred *Prediction based on a running mean*

Description

Function to predict values based on a running mean (or another function) of a numeric vector.

Usage

```
runn_mean_pred(
  indep,
  dep,
  pred,
  runn_mean = 11,
  na.rm = FALSE,
  exclude_central_value = FALSE,
  FUN = mean
)
```

Arguments

indep	numeric vector of independent variables, should be sequential
dep	numeric vector of dependent variables
pred	numeric vector of values to be predicted
runn_mean	number of vector elements to use for calculating the running mean
na.rm	ignore NA values when calculating means. Defaults to FALSE.
exclude_central_value	exclude central value in calculating means. Defaults to FALSE.
FUN	function to be applied. For a running mean, this is usually mean (the default), but other functions can also be specified here (the na.rm parameter won't work then, and the function has to be dependent on one numeric variable only.

Details

The running mean calculation that underlies the prediction is based purely on the sequence of observed values, without accounting for any variation in intervals of the independent data. This means that the function performs best with regularly spaced independent variables. Note that the function will return NA when asked to predict values that are outside the range of independent values provided as input. The prediction results are computed by linearly interpolating between the running mean values determined for the nearest neighbors of the value that is to be predicted.

Value

list of two elements, with $x containing the values to be predicted and $predicted the predicted values

Author(s)

Eike Luedeling

Examples

```
indep<-(1:100)
dep<-sin(indep/20)+rnorm(100)/5
pred<-c(12,13,51,70,90)

predicted<-runn_mean_pred(indep,dep,pred,runn_mean = 25)

plot(dep~indep)
points(predicted$predicted~predicted$x,col="red",pch=15)
```

---

save_temperature_scenarios

Save temperature scenarios generated with temperature_generation

---

Description

The temperature_generation can produce synthetic temperature scenarios, but it can take a while to run, especially for large ensembles of climate scenarios. The save_temperature_scenarios function can then save these scenarios to disk as a series of .csv files, so that they can later be used again, without re-running the generation function. Conversely, the load_temperature_scenarios function allows reading the data back into R. This function also works with any other list of data.frames.

Usage

```
save_temperature_scenarios(generated_temperatures, path, prefix)
```

Arguments

generated_temperatures	list of temperature scenarios produced with the temperature_generation function.
path	character string indicating the file path where the files are to be written.
prefix	character string specifying the prefix for all files.

Value

no values are returned, but files are written as a side_effect.

Author(s)

Eike Luedeling

Examples

```
temps<-list(Element1=data.frame(a=1,b=2),Element2=data.frame(a=c(2,3),b=c(8,4)))
# save_temperature_scenarios(temps,path=getwd(),prefix="temperatures")
# temps_reloaded<-load_temperature_scenarios(path=getwd(),prefix="temperatures")
```

`select_by_file_extension` *Select string that end in a particular way (e.g. a certain file extension)*

Description

Sometimes it makes sense to apply a function to several files in a folder, but only to those of a particular file type. This function can selects all elements in a vector of strings that end in a particular way, e.g. on a common file extension.

Usage

```
select_by_file_extension(strings, file_extension)
```

Arguments

`strings`	vector of character strings for elements to be extracted from.
`file_extension`	character string specifying the extension of the file type to be selected. This can also be any other trailing string that marks all vector elements to be selected.

Value

subset of the strings vector that only contains the elements that end on file_extension.

Author(s)

Eike Luedeling

Examples

```
select_by_file_extension(c("Temp1.csv","Temp1.xls","Temp2.csv","Temp2.xls"),"csv")
select_by_file_extension(c("red car","blue car","yellow duck"), "car")
```

`stack_hourly_temps` *Stacking of hourly temperatures*

Description

This function processes hourly temperatures generated by make_hourly_temps for calculation of chilling and forcing. The chilling function requires temperatures to be in a long list, and this function prepares them in this way.

Usage

```
stack_hourly_temps(
  weather = NULL,
  latitude = 50,
  hour_file = NULL,
  keep_sunrise_sunset = FALSE
)
```

Arguments

`weather`	weather data frame containing either daily minimum ("Tmin") and maximum ("Tmax") temperatures in the format generated by fix_weather, of hourly temperatures in the format generated by make_hourly_temps (see below; this can also be passed as hour_file).
`latitude`	the geographic latitude (in decimal degrees) of the location of interest
`hour_file`	this is a data frame of hourly temperatures, as generated by make_hourly_temps. It has columns describing the date (Year+JDay or Year+Month+Day) and 24 columns called Hour_1 ... Hour_24 that contain hourly temperatures. This is no longer required, since weather can be specified by the weather argument. This parameter is only for compatibility with earlier versions of chillR.
`keep_sunrise_sunset`	boolean variable indicating whether information on sunrise, sunset and daylength, which is calculated for producing hourly temperature records, should be preserved in the output. Defaults to FALSE.

Value

list containing two elements: hourtemps: data frame containing all the columns of the input data frame, except the hourly temperatures. Instead, two columns are added: Hour is the hour of the day, and Temp is the corresponding modeled mean temperature for that hour. QC: either the Quality control attribute ("QC") passed into the function within the daily temperature record produced by fix_weather, or NA.

Author(s)

Eike Luedeling

References

Luedeling E, Kunz A and Blanke M, 2013. Identification of chilling and heat requirements of cherry trees - a statistical approach. International Journal of Biometeorology 57,679-689.

Examples

```
weather<-fix_weather(KA_weather[which(KA_weather$Year>2004),])

hourtemps<-stack_hourly_temps(weather, latitude=50.4)
```

---

stage_transitions *Compute what it takes to advance through development stages*

Description

Function to compute the thermal requirements of transitioning through a series of developmental stages.

Usage

```
stage_transitions(
  observations,
  hourtemps,
  stages,
  models = list(Chill_Portions = Dynamic_Model, GDH = GDH),
  max_steps = length(stages)
)
```

Arguments

observations	data.frame containing observed developmental dates, e.g. different stages of flower or leaf development. Should contain the columns 'Stage' (containing the names of the development stages), 'Season' (containing the 'development year' the observation belongs to, e.g. budbreak for trees may be considered a stage of the 'dormancy year' that started in the previous calendar year), 'Year' (the calendar year the observation was made), 'JDay' (the Julian Date, a.k.a. day of the year, that the stage was observed).
hourtemps	a list of two elements, with element 'hourtemps' being a dataframe of hourly temperatures (e.g. produced by stack_hourly_temps). This data frame must have a column for Year, a column for JDay (Julian date, or day of the year), a column for Hour and a column for Temp (hourly temperature). The second (optional) element is QC, which is a data.frame indicating completeness of the dataset. This is automatically produced by stack_hourly_temps. This also works if only the 'hourtemps' dataframe is passed to the function.

stages	character vector containing the relevant development stages in their order of occurrence.
models	named list of models that should be applied to the hourly temperature data. These should be functions that take as input a vector of hourly temperatures. This defaults to list(Chill_Portions=Dynamic_Model, GDH=GDH), models that are often used for describing chill and heat accumulation in temperate fruit trees.
max_steps	integer indicating the maximum number of stage steps (i.e. transitions from one step to the next), for which thermal requirements should be calculated. This defaults to length(stages), which is also the maximum value. If only requirements between each stage and the following stage are of interest, this should be set to 1.

Value

data frame with rows for all transitions that occurred during the observed records and the values of the metrics specified in 'models' that accrued between the respective dates. Columns are c('Season','Stage', 'to_Stage','stage_steps') and one column for each thermal metrics.

Author(s)

Eike Luedeling

Examples

```
hourtemps<-stack_hourly_temps(KA_weather)
observations<-data.frame(Stage=c("V1","V2","V3","V1","V2","V3","V1","V3"),
                         Season=c(2001,2001,2001,2002,2002,2002,2003,2003),
                         Year=c(2001,2001,2001,2002,2002,2002,2003,2003),
                         JDay=c(30,45,60,35,42,55,37,62))
stages<-c("V1","V2","V3")

stage_transitions(observations,hourtemps,stages)
```

StepChill_Wrapper *StepChill_Wrapper*

Description

Same as UniChill_Wrapper, but with a step function for chilling

Usage

```
StepChill_Wrapper(x, par)
```

Arguments

x	data.frame with at least columns 'Temp' and 'JDay'
par	numeric vector of length 7 with the parameters of the StepChill model: 1. Tc, 2. bf, 3. cf, 4. Cstar and 5. Fstar.

Value

A single numeric value with the JDay prediction for the temperaturs in 'x$Temp' and the model parameters in 'par'.

Author(s)

Carsten Urbach <urbach@hiskp.uni-bonn.de>

References

Isabelle Chuine, A Unified Model for Budburst of Trees, J. theor. Biol. (2000) 207

Asse et al., Process-based models outcompete correlative models in projecting spring phenology of trees in a future warmer climate,Agricultural and Forest Meteorology (2020) 107913

---

step_model *Calculation of cumulative temperature metric according to a user-defined stepwise weight function*

Description

This function calculates heat for temperate trees according to a stepwise model provided by the user.

Usage

```
step_model(
  HourTemp,
 df = data.frame(lower = c(-1000, 1.4, 2.4, 9.1, 12.4, 15.9, 18), upper = c(1.4, 2.4,
    9.1, 12.4, 15.9, 18, 1000), weight = c(0, 0.5, 1, 0.5, 0, -0.5, -1)),
  summ = TRUE
)
```

Arguments

HourTemp	Vector of hourly temperatures.
df	data.frame with three columns: lower, upper and weight. lower should contain the lower boundary of a chilling weight interval and upper should contain the upper boundary. weight indicates the weighting to be applied to the respective temperature interval.
summ	Boolean parameter indicating whether calculated metrics should be provided as cumulative values over the entire record (TRUE) or as the actual accumulation for each hour (FALSE).

Details

Temperature-based metric calculated according to the user-defined model.

Value

Vector of length length(HourTemp) containing the cumulative temperature metric over the entire duration of HourTemp.

Author(s)

Eike Luedeling

Examples

```
weather<-fix_weather(KA_weather[which(KA_weather$Year>2006),])

stack<-stack_hourly_temps(weather,latitude=50.4)

df=data.frame(
  lower=c(-1000,1,2,3,4,5,6),
  upper=c(1,2,3,4,5,6,1000),
  weight=c(0,1,2,3,2,1,0))

custom<-function(x) step_model(x,df)

custom(stack$Temp)

models<-list(Chilling_Hours=Chilling_Hours,Utah_Chill_Units=Utah_Model,
Chill_Portions=Dynamic_Model,GDH=GDH,custom=custom)

tempResponse(stack,Start_JDay = 305,End_JDay = 60,models)
```

---

summary.bootstrap_phenologyFit

summary.bootstrap_phenologyFit

Description

Summarise a ‘bootstrap_phenologyFit‘ object

Usage

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

Arguments

object	class 'bootstrap_phenologyFit' to summarise
...	generic parameters, ignored here

Value

No return value.

---

summary.phenologyFit *summary phenologyFit*

Description

summary phenologyFit

Usage

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

Arguments

object	class phenologyFit. object to summarise
...	additional parameters, ignored here

Value

No return value.

---

temperature_generation

Generation of synthetic temperature records

Description

Function to incorporate the temperature generation function of the RMAWGEN weather generator into chillR. The weather generator is calibrated using the weather data.frame (years between years[1] and years[2]), and then generates synthetic weather for a user-defined time frame (bounded by sim_years[1] and sim_years[2]). Monthly change vectors for minimum and maximum temperatures can be specified to allow generation of temperature change scenarios.

Usage

```
temperature_generation(
  weather,
  years,
  sim_years,
  temperature_scenario = data.frame(Tmin = rep(0, 12), Tmax = rep(0, 12)),
  seed = 99,
  check_temperature_scenario_type = TRUE,
  temperature_check_args = NULL,
  max_reference_year_difference = 5,
  warn_me = TRUE,
  remove_NA_scenarios = TRUE
)
```

Arguments

weather	daily weather, as produced with the fix_weather function. Can also be generated by other means, but should contain the columns c("Month","Day","Year","Tmin","Tmax").
years	vector of length 2 indicating the start and end year of the time interval to be used for calibrating the temperature generator.
sim_years	vector of length 2 indicating the start and end year of the time interval for which temperatures are to be generated.
temperature_scenario	can be one of three options: 1) a data.frame with two columns Tmin and Tmax and n_intervals (default: 12) rows containing temperature changes for all time intervals, or absolute temperatures for these intervals. 2) a temperature scenario object, consisting of the following elements: 'data' = a data frame with n_intervals elements containing the absolute or relative temperature information (as in input option 1); 'scenario_year' = the year the scenario is representative of; 'reference_year' = the year the scenario is representative of; 'scenario_type' = the scenario type ('absolute' or 'relative' - if NA, this is assigned automatically); 'labels' = and elements attached to the input temperature_scenario as an element names 'labels'. A subset of these elements can also be specified, but 'data' must be present. 3) a (named or unnamed) list containing multiple objects of types 1 and 2. In this case, outputs are generated for all scenarios.
seed	integer specifying the random seed for the weather generation.
check_temperature_scenario_type	boolean variable specifying whether temperature scenarios should be checked - and the scenario_type updated if necessary - with the check_temperature_scenario function.
temperature_check_args	list of arguments to be passed to the check_temperature_scenario function. Check documentation of that function for details.
max_reference_year_difference	for relative temperature scenarios, the maximum difference between the reference years of the scenario and the weather record used for calibration (the median of the two elements in the 'years' argument.

warn_me boolean variable specifying whether warnings should be shown. Defaults to TRUE.

remove_NA_scenarios
boolean parameter indicating whether temperature scenarios that contain NA values should be removed. Such scenarios would generate an error.

Details

Note that this function uses the temperature generation algorithms of the RMAWGEN package. For more details, refer to the documentation of this package.

Value

list of data.frames containing the simulated weather, with columns c("YEARMODA", "DATE","Year","Month","Day","Tmin
If temperature_scenario is a list, the output list contains simulated temperature records for all scenarios.

Author(s)

Eike Luedeling

Examples

```
## Examples are #d out to pass CRAN checks. Remove #s to run them.
# Temp<-temperature_generation(KA_weather,years=c(1999,2001),
#   sim_years = c(2001,2002),temperature_scenario = data.frame(Tmin=c(1,3,2,1,5,7,3,2,1,5,4,3),
#    Tmax=c(1,2,3,2,1,3,2,1,2,3,4,5)))

# Temp<-temperature_generation(weather=KA_weather,years=c(1999,2001),
#  sim_years = c(2005,2006),
#   temperature_scenario=data.frame(Tmin=c(1,3,5,8,12,15,15,15,10,8,3,1),
#    Tmax=c(6,8,10,13,17,20,20,20,15,13,8,6)))
```

---

temperature_scenario_baseline_adjustment

Make temperature scenario relative to a particular baseline

Description

When interpreting future (or past) temperature scenarios that provide absolute temperatures, it is important to consider the temperature baseline, i.e. a temperature scenario produced with similar models and methods that corresponds to the current temperature regime. Such baselines are normally available from the same source that provided the future scenarios. This function implements this adjustment. The function can be used for two situations: 1) two absolute temperature scenarios: the output is the difference between the scenarios, i.e. a relative temperature scenario describing the difference between monthly temperature extreme means between the two

scenarios. 2) two relative temperature scenarios: the output is a relative temperature scenario that describes the difference between the scenario year of the temperature_scenario and the baseline year of the baseline_temperature_scenario. This only works if the scenario_year of the baseline_temperature_scenario is the same as the reference_year of the temperature_scenario.

Usage

```
temperature_scenario_baseline_adjustment(
  baseline_temperature_scenario,
  temperature_scenario,
  temperature_check_args = NULL,
  warn_me = TRUE,
  required_variables = c("Tmin", "Tmax")
)
```

Arguments

`baseline_temperature_scenario`
: baseline temperature scenario (e.g. produced with 'extract_temperatures_from_grids'). This is a temperature scenario object, consisting of the following elements: 'data' = data.frame with two columns Tmin and Tmax containing absolute (normally monthly) mean minimum and maximum temperatures; 'reference_year' = the year the scenario refers to (this is normally NA for absolute temperature scenarios, because they don't require considering a reference scenario); 'scenario_type' = the scenario type, normally "absolute" (but can also be "relative" or NA - then the type is automatically assigned); 'labels' = elements attached to the input temperature_scenario. A subset of these elements can also be specified, but 'data' must be present.

`temperature_scenario`
: can be one of three options: 1) a data.frame with two columns Tmin and Tmax and n_intervals (default: 12) rows containing temperature changes for all time intervals, or absolute temperatures for these intervals. 2) a temperature scenario object, consisting of the following elements: 'data' = a data frame with n_intervals elements containing the absolute or relative temperature information (as in input option 1); 'scenario_year' = the year the scenario is representative of; 'reference_year' = the year the scenario is representative of; 'scenario_type' = the scenario type ('absolute' or 'relative' - if NA, this is assigned automatically); 'labels' = and elements attached to the input temperature_scenario as an element names 'labels'. A subset of these elements can also be specified, but 'data' must be present. 3) a list of elements of type 1 or 2. Then the adjustment is done for all elements.

`temperature_check_args`
: list of arguments to be passed to the check_temperature_scenario function. Check documentation of that function for details.

`warn_me`
: boolean variable specifying whether warnings should be shown. Defaults to TRUE.

`required_variables`
: character vectors containing names of variables that must be included in the scenario.

Value

temperature scenario object, consisting of the following elements: 'data' = a data frame with n_intervals elements containing the absolute or relative temperature information. 'reference_year' = the year the scenario is representative of. 'scenario_type' = the scenario type ('absolute' or 'relative'); 'labels' = and elements attached to the input temperature_scenario as an element names 'labels'.

The function also returns warnings, where elements are missing or the scenario_type appears to be wrong, and it stops with an error, if the scenario isn't specified in a format that is usable by chillR.

Author(s)

Eike Luedeling

Examples

```
baseline_temperature_scenario<-list(data=data.frame(Tmin=c(1,1,1,1,1,1,1,1,1,1,1,1),
                                                    Tmax=c(1,1,1,1,1,1,1,1,1,1,1,1)),
                                                    scenario_year=1990,
                                                    reference_year=1975,
                                                    scenario_type="relative")

temperature_scenario<-list(data=data.frame(Tmin=c(4,4,4,4,4,4,4,4,4,4,4,4),
                                           Tmax=c(4,4,4,4,4,4,4,4,4,4,4,4)),
                                           scenario_year=2000,
                                           reference_year=1990,
                                           scenario_type="relative")

relative_temperature_scenario<-temperature_scenario_baseline_adjustment(
                     baseline_temperature_scenario,temperature_scenario,
                     temperature_check_args=NULL)

baseline_temperature_scenario<-list(data=data.frame(Tmin=c(-5,-2,2,5,10,12,15,15,12,10,5,1),
                                              Tmax=c( 1, 4,7,10,15,18,22,24,17,15,11,6)),
                                                    scenario_year=1980,
                                                    reference_year=NA,
                                                    scenario_type="absolute")

temperature_scenario<-list(data=data.frame(Tmin=c(-3,0,4,7,12,14,17,17,14,12,7,3),
                                              Tmax=c(3,6,9,12,17,20,24,26,19,17,13,8)),
                                           scenario_year=2000,
                                           reference_year=NA,
                                           scenario_type="absolute")

relative_temperature_scenario<-temperature_scenario_baseline_adjustment(
                     baseline_temperature_scenario,temperature_scenario,
                     temperature_check_args=NULL)
```

`temperature_scenario_from_records` *Make monthly temperature scenario from historic records*

Description

Produces a list of scenarios containing monthly means for Tmin and Tmax that are representative of particular years. These scenario are computed by applying linear regression to a file containing Tmin and Tmax records, and using the regression model to calculate typical values for the user-specified years.

Usage

```
temperature_scenario_from_records(
  weather,
  year,
  weather_start = NA,
  weather_end = NA,
  scen_type = "running_mean",
  runn_mean = 15
)
```

Arguments

`weather`	daily weather, as produced with the fix_weather function. Can also be generated by other means, but shouold contain the columns c("Month","Day","Year","Tmin","Tmax").
`year`	numeric vector of years, for which the scenario is to be produced.
`weather_start`	start year of the period to be considered in calculating the regression. Defaults to NA, which means the first year of the record is used as start year.
`weather_end`	end year of the period to be considered in calculating the regression. Defaults to NA, which means the last year of the record is used as end year.
`scen_type`	character string, either "regression" or "running_mean", specifying how the scenario should be produced. "regression" computed the scenario based on an assumed linear trend in the data; "running_mean" uses a running mean function instead, with the length of the running mean window determined by the runn_mean parameter. The default is a running mean function, since the assumption of a linear trend often does not hold.
`runn_mean`	number of vector elements to use for calculating the running mean; this is reduced, if the time series is not long enough to accommodate the specified window. Defaults to 15.

Details

This function produces outputs that can be used as input for the temperature_generation function. Sample applications are the use of the temperature_generation function for making replicate

weather records for a given year for risk assessment purposes, or the generation of a weather scenario that can be compared with other datasets (e.g. climate scenarios based on the WorldClim dataset refer to a 1951-2000 baseline, so that meaningful use of such scenarios for local contexts requires consideration of a scenario that corresponds to temperatures in 1975, the central year of this period).

Value

list of climate scenario objects, consisting of the following elements: 'data' = a data frame with n_intervals elements containing the absolute temperature information. 'scenario_year' = the year the scenario is representative of, i.e. the specified 'year' parameter. 'reference_year' = NA (because this is an absolute temperature scenarios, not a relative one); 'scenario_type' = 'absolute' (because this is an absolute temperature scenario, not a relative one); 'labels' = 'regression-based scenario'.

Author(s)

Eike Luedeling

Examples

```
temperature_scenario_from_records(weather=KA_weather,year=2001,weather_start=2000,weather_end=2005)
```

---

tempResponse *Calculation of climatic metrics from hourly temperature records*

Description

Extension of the chilling function, which calculated four pre-defined temperature-based metrics. This function has more flexibility, because it allows specifying the models that should be calculated. These can be selected from a small set of models provided with chillR, but they can also be defined by the user. Precondition at the moment is that they require hourly temperature only as inputs.

Usage

```
tempResponse(
  hourtemps,
  Start_JDay = 1,
  End_JDay = 366,
  models = list(Chilling_Hours = Chilling_Hours, Utah_Chill_Units = Utah_Model,
    Chill_Portions = Dynamic_Model, GDH = GDH),
  misstolerance = 50,
  whole_record = FALSE,
  mean_out = FALSE
)
```

Arguments

hourtemps	a list of two elements, with element 'hourtemps' being a dataframe of hourly temperatures (e.g. produced by stack_hourly_temps). This data frame must have a column for Year, a column for JDay (Julian date, or day of the year), a column for Hour and a column for Temp (hourly temperature). The second (optional) element is QC, which is a data.frame indicating completeness of the dataset. This is automatically produced by stack_hourly_temps.
Start_JDay	the start date (in Julian date, or day of the year) of the period, for which chill and heat should be quantified.
End_JDay	the end date (in Julian date, or day of the year) of the period, for which chill and heat should be quantified.
models	named list of models that should be applied to the hourly temperature data. These should be functions that take as input a vector of hourly temperatures. This defaults to the set of models provided by the chilling function.
misstolerance	maximum percentage of values for a given season that can be missing without the record being removed from the output. Defaults to 50.
whole_record	boolean parameter indicating whether the metrics should be summed over the entire temperature record. If set to TRUE (default is FALSE), then the function ignores the specified start and end dates and simply returns the totals of each metric that accumulated over the entire temperature record.
mean_out	boolean parameter indicating whether the mean of the input metric (e.g. temperature) should be returned in a column named "Input_mean".

Details

The function calculates the total of user-specified temperature-based metrics over periods delineated by Start_JDay and End_JDay. Models for calculating these metrics are provided in the models list, whose elements are named functions that convert hourly temperature records into a cumulative record of the climate metric of interest. The metric is then added up cumulatively over the entire temperature record and then summarized by season. Examples of functions that can be used are Chilling_Hours, Utah_Model, Dynamic_Model and GDH. The custom_model function allows customized simply weight-based models, which assign differential weights to temperatures within certain intervals. See custom_model documentation for details.

Value

data frame showing totals for all specified models for the respective periods for all seasons included in the temperature records. Columns are Season, End_year (the year when the period ended) and Days (the duration of the period), as well as one column per model, which receives the same name as the function in the models list. If the weather input consisted of a list with elements stack and QC, the output also contains columns from QC that indicate the completeness of the weather record that the calculations are based on.

Author(s)

Eike Luedeling

References

The chillR package:

Luedeling E, Kunz A and Blanke M, 2013. Identification of chilling and heat requirements of cherry trees - a statistical approach. International Journal of Biometeorology 57,679-689.

Examples

```
weather<-fix_weather(KA_weather[which(KA_weather$Year>2006),])

hourtemps<-stack_hourly_temps(weather,latitude=50.4)

df=data.frame(
  lower=c(-1000,1,2,3,4,5,6),
  upper=c(1,2,3,4,5,6,1000),
  weight=c(0,1,2,3,2,1,0))

custom<-function(x) step_model(x,df)

models<-list(Chilling_Hours=Chilling_Hours,Utah_Chill_Units=Utah_Model,Chill_Portions=
  Dynamic_Model,GDH=GDH,custom=custom)

tempResponse(hourtemps,Start_JDay = 305,End_JDay = 60,models)
```

---

tempResponse_daily_list

Calculation of climatic metrics from lists of daily temperature records

---

Description

Wrapper for the tempResponse function, to facilitate its use on lists of daily temperature records, e.g. those produced by the `temperature_generation` function. Daily temperature records are converted into hourly records using either the `stack_hourly_temps` function or an empirical relationship between observed hourly temperatures and daily temperature extremes (see `Empirical_hourly_temperatures` for details). These hourly records are then used as input into the `tempResponse` function, to which most parameters are passed. See the documentation of `tempResponse` for more details.

Usage

```
tempResponse_daily_list(
  temperature_list,
  latitude,
  Start_JDay = 1,
  End_JDay = 366,
  models = list(Chilling_Hours = Chilling_Hours, Utah_Chill_Units = Utah_Model,
    Chill_Portions = Dynamic_Model, GDH = GDH),
  misstolerance = 50,
```

```
  whole_record = FALSE,
  empirical = NULL,
  mean_out = FALSE
)
```

Arguments

temperature_list	list of daily temperature records, as produced by temperature_generation.
latitude	latitude of the location of interest (used for generating hourly records).
Start_JDay	the start date (in Julian date, or day of the year) of the period, for which chill and heat should be quantified.
End_JDay	the end date (in Julian date, or day of the year) of the period, for which chill and heat should be quantified.
models	named list of models that should be applied to the hourly temperature data. These should be functions that take as input a vector of hourly temperatures. This defaults to the set of models provided by the chilling function.
misstolerance	maximum percentage of values for a given season that can be missing without the record being removed from the output. Defaults to 50.
whole_record	boolean parameter indicating whether the metrics should be summed over the entire temperature record. If set to TRUE (default is FALSE), then the function ignores the specified start and end dates and simply returns the totals of each metric that accumulated over the entire temperature record.
empirical	indicates whether hourly temperatures should be generated based on an idealized temperature curve (set to NULL, the default) or an empirically derived relationship between hourly temperatures and daily temperature extremes (see Empirical_hourly_temperatures and Empirical_daily_temperature_curve, also for the format of the empirical prediction coefficient data.frame). If the latter, this parameter needs to be a data.frame including columns Month, Hour and Prediction_coefficients. See Empirical_daily_temperature_curve for further details on the format.
mean_out	boolean parameter indicating whether the mean of the input metric (e.g. temperature) should be returned in a column named "Input_mean".

Value

data frame showing totals for all specified models for the respective periods for all seasons included in the temperature records. Columns are Season, End_year (the year when the period ended) and Days (the duration of the period), as well as one column per model, which receives the same name as the function in the models list. If the weather input consisted of a list with elements stack and QC, the output also contains columns from QC that indicate the completeness of the weather record that the calculations are based on.

Author(s)

Eike Luedeling

References

The chillR package:

Luedeling E, Kunz A and Blanke M, 2013. Identification of chilling and heat requirements of cherry trees - a statistical approach. International Journal of Biometeorology 57,679-689.

Examples

```
weather<-fix_weather(KA_weather[which(KA_weather$Year>2006),])
temperature_list<-list(weather,weather,weather)

tempResponse_daily_list(temperature_list,latitude=50.4)
```

---

tempResponse_hourtable

Add metric accumulation to table of hourly temperatures

Description

This function calculates cumulative values for temperature response metrics for every hour of an hourly temperature record. The count is restarted on a specified date each year. The function is a generalized version of chilling_hourtable, which only worked with three predefined chilling one predefined heat metrics.

Usage

```
tempResponse_hourtable(
  hourtemps,
  Start_JDay,
  models = c(Chill_Portions = Dynamic_Model, GDH = GDH_model)
)
```

Arguments

hourtemps	a dataframe of stacked hourly temperatures (e.g. produced by stack_hourly_temps). This data frame must have a column for Year, a column for JDay (Julian date, or day of the year), a column for Hour and a column for Temp (hourly temperature).
Start_JDay	the start date (in Julian date, or day of the year) of the calculation for the four metrics. The count is restarted on this date every year.
models	named list of models that should be applied to the hourly temperature data. These should be functions that take as input a vector of hourly temperatures. This defaults to c(Chill_Portions = Dynamic_Model, GDH = GDH_model), which refer to the Dynamic chill model and the Growing Degree Hours model functions contained in chillR.

Value

data frame consisting of all the columns of the THourly input data frame, plus one additional column for each model, which contains the cumulative number of model metrics since the last Start_JDay).

Note

After doing extensive model comparisons, and reviewing a lot of relevant literature, I do not recommend using the Chilling Hours or Utah Models, especially in warm climates! The Dynamic Model (Chill Portions), though far from perfect, seems much more reliable.

Author(s)

Eike Luedeling

References

Model references:

Dynamic Model:

Erez A, Fishman S, Linsley-Noakes GC, Allan P (1990) The dynamic model for rest completion in peach buds. Acta Hortic 276, 165-174

Fishman S, Erez A, Couvillon GA (1987a) The temperature dependence of dormancy breaking in plants - computer simulation of processes studied under controlled temperatures. J Theor Biol 126(3), 309-321

Fishman S, Erez A, Couvillon GA (1987b) The temperature dependence of dormancy breaking in plants - mathematical analysis of a two-step model involving a cooperative transition. J Theor Biol 124(4), 473-483

Growing Degree Hours:

Anderson JL, Richardson EA, Kesner CD (1986) Validation of chill unit and flower bud phenology models for 'Montmorency' sour cherry. Acta Hortic 184, 71-78

Review on chilling models in a climate change context:

Luedeling E, 2012. Climate change impacts on winter chill for temperate fruit and nut production: a review. Scientia Horticulturae 144, 218-229

Examples

```
weather<-fix_weather(KA_weather[which(KA_weather$Year>2008),])

hourtemps<-stack_hourly_temps(weather,latitude=50.4)

cht<-chilling_hourtable(hourtemps,20)
```

test_if_equal *Test if all character vectors in a string are equal*

Description

Compares all elements of a vector of numbers or character strings and returns TRUE if they are all the same, FALSE otherwise.

Usage

```
test_if_equal(test_vector)
```

Arguments

test_vector	vector of strings or numbers to be tested.

Value

TRUE if all elements of the vector are the same; FALSE otherwise.

Author(s)

Eike Luedeling

Examples

```
test_if_equal(c(1,3,1))
test_if_equal(c("a","a","a"))
test_if_equal(c("a","b","a"))
```

UniChill_Wrapper *UniChill_Wrapper*

Description

UniChill_Wrapper

Usage

```
UniChill_Wrapper(x, par)
```

Arguments

x	data.frame with at least columns 'Temp' and 'JDay'
par	numeric vector of length 7 with the parameters of the UniChill model: 1. ac, 2. bc, 3. cc, 4. bf, 5. cf, 6. Cstar and 7. Fstar.

Value

A single numeric value with the JDay prediction for the temperaturs in ‘x$Temp‘ and the model parameters in ‘par‘.

Author(s)

Carsten Urbach <urbach@hiskp.uni-bonn.de>

References

Isabelle Chuine, A Unified Model for Budburst of Trees, J. theor. Biol. (2000) 207

`UnifiedModel_Wrapper` *UnifiedModel_Wrapper*

Description

UnifiedModel_Wrapper

Usage

```
UnifiedModel_Wrapper(x, par)
```

Arguments

`x`	data.frame with at least columns ‘Temp‘ and ‘JDay‘
`par`	numeric vector of length 9 with the parameters of the unified model: 1. ac, 2. bc, 3. cc, 4. bf, 5. cf, 6. w, 7. k, 8. Cstar and 9. tc.

Value

A single numeric value with the JDay prediction for the temperaturs in ‘x$Temp‘ and the Unified Model parameters in ‘par‘.

Author(s)

Carsten Urbach <urbach@hiskp.uni-bonn.de>

References

Isabelle Chuine, A Unified Model for Budburst of Trees, J. theor. Biol. (2000) 207

UniForce_Wrapper *UniForce_Wrapper*

Description

UniForce_Wrapper

Usage

```
UniForce_Wrapper(x, par)
```

Arguments

x	data.frame with at least columns 'Temp' and 'JDay'
par	numeric vector of length 4 with the parameters of the UniForce model: 1. bf, 2. cf, 3. Fstar, 4. t1.

Value

A single numeric value with the JDay prediction for the temperaturs in 'x$Temp' and the Unified Model parameters in 'par'.

Author(s)

Carsten Urbach <urbach@hiskp.uni-bonn.de>

References

Isabelle Chuine, A Unified Model for Budburst of Trees, J. theor. Biol. (2000) 207

Utah_Model *Calculation of cumulative chill according to the Utah Model*

Description

This function calculates winter chill for temperate trees according to the Utah Model.

Usage

```
Utah_Model(HourTemp, summ = TRUE)
```

Arguments

HourTemp	Vector of hourly temperatures.
summ	Boolean parameter indicating whether calculated metrics should be provided as cumulative values over the entire record (TRUE) or as the actual accumulation for each hour (FALSE).

Details

Units of the Utah Model are calculated as suggested by Richardson et al. (1974) (different weights for different temperature ranges, and negation of chilling by warm temperatures).

Value

Vector of length length(HourTemp) containing the cumulative Utah Chill Units over the entire duration of HourTemp.

Note

After doing extensive model comparisons, and reviewing a lot of relevant literature, I do not recommend using the Utah Model, especially in warm climates! The Dynamic Model (Chill Portions), though far from perfect, seems much more reliable.

Author(s)

Eike Luedeling

References

Utah Model reference:

Richardson EA, Seeley SD, Walker DR (1974) A model for estimating the completion of rest for Redhaven and Elberta peach trees. HortScience 9(4), 331-332

Examples

```
weather<-fix_weather(KA_weather[which(KA_weather$Year>2006),])

stack<-stack_hourly_temps(weather,latitude=50.4)

Utah_Model(stack$hourtemps$Temp)
```

VIP *Calculate VIP scores for PLS regression*

Description

This function calculates the Variable Importance in the Projection statistic for the Partial Least Squares regression. It is used in the PLS function. Executing it in isolation will probably not be useful to most users.

Usage

```
VIP(object)
```

Arguments

object	an mvr object, as produced by the pls procedure or a range of other functions

Details

This is required to produce the VIP scores for the PLS procedure.

Value

data frame with as many columns as independent variables are input into the PLS analysis. The number of columns corresponds to the number of latent components selected for the analysis. Values in the data frame are the VIP values corresponding to each variable for the respective component.

Author(s)

Eike Luedeling, but the function was mainly copied from http://mevik.net/work/software/pls.html; the reference given there is listed below

References

the function is mostly identical to the one provided on http://mevik.net/work/software/pls.html.

Here is the reference given there:

Chong, Il-Gyo & Jun, Chi-Hyuck, 2005, Performance of some variable selection methods when multicollinearity is present, Chemometrics and Intelligent Laboratory Systems 78, 103-112

This reference refers to the chillR package:

Luedeling E, Kunz A and Blanke M, 2013. Identification of chilling and heat requirements of cherry trees - a statistical approach. International Journal of Biometeorology 57,679-689.

Examples

```
PLS_results<-PLS_pheno(
  weather_data=KA_weather,
  split_month=6,   #last month in same year
  bio_data=KA_bloom,return.all=TRUE)

#return.all makes the function return the whole PLS object - needed for next line to work

VIP(PLS_results$PLS_output)
```

`weather2chillR` *Convert downloaded weather to chillR format*

Description

Convert downloaded weather data into a data frame that makes running other chillR functions easy.

Usage

```
weather2chillR(downloaded_weather, database = "GSOD", drop_most = TRUE)
```

Arguments

`downloaded_weather`	weather file downloaded with the get_weather function. This can be a data.frame or a list with elements database and weather as produced by get_weather
`database`	weather database that the file was downloaded from. Can only be "GSOD" at this point.
`drop_most`	boolean variable indicating if most columns should be dropped from the file. If set to TRUE (default), only essential columns for running chillR functions are retained.

Details

weather databases, from which chillR can download data: NOAA NCDC Global Summary of the Day - "GSOD" (https://data.noaa.gov/dataset/global-surface-summary-of-the-day-gsod)

California Irrigation Management Information System (CIMIS) - "CIMIS" (http://www.cimis.water.ca.gov/)

University of California Integrated Pest Management (UCIPM) - "UCIPM" (http://ipm.ucdavis.edu/WEATHER/)

data should first be downloaded with get_weather. Then the database name is passed to the function and can be skipped in the call. If only a data.frame is provided, then the database name must be specified.

Processing the data with this function will make the data work well with the remainder of this package.

Value

a data.frame with weather data, according to the downloaded file provided as input. If drop_most is FALSE, all columns from the original dataset are preserved, although some column names are adjusted to chillR's preferences ("Year","Month","Day","Tmin","Tmax","Tmean","Prec"). If drop_most is TRUE, only columns likely to be of interest to chillR users are retained. If a list with elements database and weather is passed to this function, this structure will be retained in the output.

Note

Many databases have data quality flags, which may sometimes indicate that data aren't reliable. These are not considered by this function!

Author(s)

Eike Luedeling

References

The chillR package:

Luedeling E, Kunz A and Blanke M, 2013. Identification of chilling and heat requirements of cherry trees - a statistical approach. International Journal of Biometeorology 57,679-689.

Examples

```
# All examples are disabled, because the database is sometimes unavailable. This then generates
# an error when R runs its package functionality checks. To run the examples, remove the # mark,
# before running the code.

#stat_list<-get_weather(location=c(lat=40,lon=-120,ele=150),time_interval=c(2015,2016),
#database="UCIPM")
#chillRcode<-stat_list[which(stat_list$Perc_interval_covered==
#max(stat_list$Perc_interval_covered)),"chillR_code"][1]
#chillRcode should equal "DOYLE.C" now.
#gw<-get_weather(location="DOYLE.C",time_interval=c(2002,2002),database="UCIPM")
#weather<-weather2chillR(gw$weather,"GSOD")
#weather<-weather2chillR(gw)
```

Winters_hours_gaps *Hourly temperature data sample*

Description

Hourly temperature data recorded in a walnut orchard near the city of Winters, California, USA for 3rd March to 11th November 2008. The dataset contains the full record of recorded temperatures, as well as an additional dataset, in which 500 data gaps of different length were introduced.

Format

A data frame with observations on the following 5 variables.

Year a numeric vector - the observation year

Month a numeric vector - the observation month

Day a numeric vector - the observation day

Hour a numeric vector - the observation day

Temp_gaps a numeric vector - daily maximum temperature

Temp a numeric vector - daily minimum temperature

Source

data were collected by Eike Luedeling, at that time at the University of California Davis (now University of Bonn) in a walnut orchard near Winters, California

Examples

```
data(Winters_hours_gaps)
```

`YEARMODA2Date` *YEARMODA to Date conversion*

Description

Converts dates in YEARMODA format to R date format

Usage

```
YEARMODA2Date(YEARMODA)
```

Arguments

`YEARMODA`	Date in YEARMODA format (e.g. 20160206 for 6th Feb 2016)

Details

Converts YEARMODA to R date

Value

Date object

Author(s)

Eike Luedeling

Examples

```
YEARMODA2Date(20001205)
YEARMODA2Date(19901003)
```

Index