Package 'LarsChill'

June 26, 2025

Type Package

Title Functions supplementing the chillR package

Version 0.0.3

RoxygenNote 7.3.2

Encoding UTF-8

License GPL-3

Description Some extra functions which are intended to enhance chillR package on the long run. Hopefully they all work...

Maintainer Lars Caspersen <lars.caspersen@uni-bonn.de>

Roxygen list(markdown = TRUE)

Depends assertthat, CFtime, dplyr, ecmwfr, graphics, magrittr, ncdf4, patchwork, R (>= 3.5), utils

Imports chillR, ggplot2, lubridate, nleqslv, purrr, readr, reshape2, RMAWGEN, Rsolnp, stats, tidyr

LazyData true

Author Lars Caspersen [aut, cre]

Contents

`convert_parameters` *Converts model parameters to standard PhenoFlex format*

Description

Takes parameters, including intermediate parameters of chill submodel and transform them to the typical PhenoFlex format.

Usage

```
convert_parameters(par, failure_return = "MEIGO")
```

Arguments

`par`	vector of length 12 with the parameters in the following order: yc, zc, s1, Tu, theta_star, theta_c, tau, pie_c, Tf, Tc, Tb, slope
`failure_return`	character, by default set to "MEIGO". Decides what should be returned if the conversion failed. In case of "MEIGO" it returns a MEIGO-compatible output, indicating that the current parameters are not suitable. If set equal "Ignore Error" it will return the best estimate. If set equal "NA" it will return NA instead.

Details

The changed parameters are theta_star, theta_c, tau and pie_c, which get converted to E0, E1, A0 and A1. In case of A0 and A1 two conversion functions are used. For E0 and E1 a nonlinear system is solved using the function nleqslv.

The conversion follows the approach documented in Fishman et al (1987) and Egea et al. (2021). For more details consult equations 5 - 8 in Egea et al (2021)

Value

vector of length 12, with the PhenoFlex parameters yc, zc, s1, Tu, E0, E1, A0, A1, Tf, Tc, Tb, slope with given temperature data and model parameters.

Author(s)

Lars Caspersen, <lars.caspersen@uni-bonn.de>

Examples

```
## Not run:
par_old <-   c(40, 190, 0.5, 25, 279, 286.1, 47.7, 28, 4, 36, 4, 1.60)
par_new <- convert_parameters(par)

## End(Not run)
```

`convert_parameters_old_to_new`

Converts parameters from old to new format

Description

Takes the standard parameters of PhenoFlex and replaces E0 to A1 parameter with intermediate parameter value.

Usage

```
convert_parameters_old_to_new(par)
```

Arguments

`par`	vector of length 12 with the parameters in the following order: yc, zc, s1, Tu, E0, E1, A0, A1, Tf, Tc, Tb, slope

Details

Original parameters include E0, E1, A0, A1 and convert those to theta_star, theta_c, tau and pie_c. Parameters theta_star and tau depend on approximated intermediate variable sigma. Pie_c is approximated using theta_star, theta_c and tau. Since the conversion is an numerical approximation, the final results differ slightly from the "true" value. This can be seen when first using `convert_parameters` and then convert the parameter again to the original format using this function. The error should be negligible. Approximation is done using the function nleqslv.

The conversion follows the approach documented in Fishman et al. (1987) and Egea et al. (2021). For more details consult equations 5 - 8 in Egea et al (2021)

Value

vector of length 12, with the PhenoFlex parameters yc, zc, s1, Tu, theta_star, theta_c, tau, pie_c, Tf, Tc, Tb, slope with given temperature data and model parameters.

Author(s)

Lars Caspersen, <lars.caspersen@uni-bonn.de>

Examples

```
## Not run:
par_old <-   c(40, 190, 0.5, 25, 279, 286.1, 47.7, 28, 4, 36, 4, 1.60)
par_new <- convert_parameters(par)
par_old_again <- convert_parameters_old_to_new(par)

## End(Not run)
```

custom_essr *Optimization algorithm, taken from MEIGOR package*

Description

Optimization algorithm of enhanced scatter search. See the MEIGOR package for more details. Took the function from the package because it seems that the MEIGOR package was deprecated and it was also painful to install the package.

Usage

```
custom_essr(problem, opts = list(maxeval = NULL, maxtime = NULL), ...)
```

Arguments

problem	list containing the initial parameter values x_0, the upper bound x_u, lower bound x_L and the evaluation function. You can also add inequality restraints to the problem statement
opts	list of arguments controlling the optimization algorithm. Most important are maxeval, controlling the maximum number of evaluations before stopping the optimization, and maxtime, maximum time in seconds for the optimizer to run
...	Further arguments that get supplied to the evaluation function in the problem statement

Details

Check the MEIGOR documentation for more details

Value

returns a list containing optimized parameters, intermediate solutions, final performance score and much more.

Author(s)

Lars Caspersen, <lars.caspersen@uni-bonn.de>

custom_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. Adjusted so that it also handles precipitation

Usage

```
custom_gen_rel_change_scenario(
  downloaded_list,
  variable = c("Tmin", "Tmax"),
  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')
variable	vector with characters, specifies the variables for which the relative change scenario should be calculated. By default: c('Tmin', 'Tmax'), for minimum and maximum temperature. Also allows 'Prec' for precipitation
scenarios	numeric vector, states the future years, for which the climate change scenarios should be generated. Usually set to c(2050, 2085).
reference_period	numeric vector specifying the years to be used as the reference period. Usually set to 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', 'Prec'),
                      year_start = 2015,
                      year_end = 2100)

download_baseline_cmip6_ecmwfr(
    area = c(55, 5.5, 47, 15.1),
    model = 'AWI-CM-1-1-MR',
     variable = c('Tmin', 'Tmax', 'Prec'),
```

```
    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,
                                area =  c(52, -7, 33, 8),
                                variable = c('Tmin', 'Tmax', 'Prec'))

custom_gen_rel_change_scenario(extracted,
                        variable = c('Tmin', 'Tmax', 'Prec'),
                        scenarios = c(2050, 2085),
                        reference_period = c(1986:2014),
                        future_window_width = 30)


## End(Not run)
```

`custom_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. Can also handle precipitation now. 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

```
custom_temperature_scenario_from_records(
  weather,
  year,
  variable = c("Tmin", "Tmax"),
  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 should contain the columns c("Month", "Day", "Year"). Also needs to contain variables specified in argument 'variable'
`year`	numeric vector of years, for which the scenario is to be produced.

variable	vector with characters. Specifies for which columns the temperature scenario is generated. By default set to c('Tmin', 'Tmax') for minimum and maximum temperature. Also allows 'Prec' for precipitation.
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, Lars Caspersen

Examples

```
## Not run:
custom_temperature_scenario_from_records(weather=chillR::KA_weather,
year=2001,
weather_start=2000,
weather_end=2005)

## End(Not run)
```

`download_seasonal_forecast` *Downloads seasonal forecast from the C3S database*

Description

Starts the download request for the seasonal forecast. The seasonal forecast data comes from the Copernicus Climate Change Service (C3S). The full name of the dataset is: Seasonal forecast daily and subdaily data on single levels. You can find more information on the dataset online: https://cds.climate.copernicus.eu/datasets/seasonal-original-single-levels?tab=overview.

Usage

```
download_seasonal_forecast(
  year,
  month,
  area,
  leadtime_hour = "all",
  fname = NULL,
  download_path = getwd(),
  start_download = TRUE,
  request_env = NULL,
  data_format = "netcdf",
  day = NULL,
  originating_centre = "dwd",
  variable = "2m_temperature",
  system = "21"
)
```

Arguments

`year`	numeric or character, indicates for which years forecast data should be downloaded. Can also be a vector of years. For most data sources, earliest possible year is 1993, some also can go back to 1981. For more details check the documentation of the dataset.
`month`	numeric or character, indicates for which months forecast data should be downloaded. This indicates the start_point of the forecast. Can also be a vector of months.
`area`	numeric, of length 4. Specifies the coordinates of the area for the downloaded forecast data. Coordinates are provided in the format: max latitude, min longitude, min latitude, max longtidue
`leadtime_hour`	numeric or character, indicates for which timepoints data should be downloaded. By default is set to 'all', requesting full length of forecast. Leadtime hour indicates timepoints (in hours since first day midnight). Data is provided every six hours, so provided values need to be divisible by six. Can be a vector of hours. Maximum value depends on data source, but in most cases cover up to six months (=4416).
`fname`	character, indicates file name for the download. By default is set to NULL, so that file name is decided automatically. Automated file naming indicates most

relevant parameter specified in the download call. For more infor of automated file naming, check the details.

`download_path` character, specifies where the downloaded file should be saved. Is by default the working directory of the R session.

`start_download` logical, indicates if download should start after the request is sent. If set TRUE (default), the function will be busy until the file is downloaded from the API. Usually it takes a while until the dataset is ready for download, so in larger requests (several years, several months) it is adviced to set to FALSE. When set to FALSE, the request is sent to API and the file can be downloaded later. See in example how the download and request can be run independently from another.

`request_env` environment object, created by the download function when parameter start_download is set FALSE. Allows the download of the requested forecast data independently from the request. Can be also used in a different R session, if the request_env is saved.

`data_format` character, decides on file format. By default is set to 'netcdf'. The API also allows to download .grib files (data_format = 'grib'). But it is adviced to use 'netcdf', as the extraction function only works for netcdf files.

`day` character or numeric, indicates the day of the month for the start of the forecast. Most sources only offer the first day of the month for the start. By default set to 1, for first day of the month

`originating_centre`
character, indicates the data source. By default set to 'dwd'. See details for more options.

`variable` character, indicates what variable to download. By default set to '2m_temperature' for temperature at 2m height. Check the documentation of the dataset for more options.

`system` character, indicates the global circulation model used for the forecast. Check documentation of the dataset, or the details, to see what combination of originating_center and system are valid.

Details

The communication with the API is done with the help of the ecmwfr package. Check out the ecmwfr package: https://cran.r-project.org/web/packages/ecmwfr/index.html The package can be used to also access other datasets. For instance, the ecmwfr package is used to download climate change projections from the Coupled Model Intercomparison Project Phase 6 (CMIP6) in the chillR package.

Combination of originating center, system and maximum leadtime_hour

- ecmwf: system = c('4', '5', '51'), year_start = 1981, leadtime_hour_end = 5160, month = 1:12, day = '01'
- ukmo: system = as.character(c(12:15, 600:604)), year_start = 1993, leadtime_hour_end = 5160, month = 1:12, day = c('01', '9', '17', '25')
- meteo_france: system = as.character(5:9),year_start = 1993, leadtime_hour_end = 5160, month = 1:12, day = c('01')
- dwd: system = as.character(c(2, 21,22)),year_start = 1993, leadtime_hour_end = 4416,month = 1:12, day = c('01')
- cmcc: system = as.character(c(3, 35)), year_start = 1993, leadtime_hour_end = 4416, month = 1:12, day = c('01')

- ncep: system = as.character(c(2)),year_start = 1993,leadtime_hour_end = 5160,month = 1:12,day = as.character(1:30)
- jma: system = as.character(c(2, 3)),year_start = 1981,leadtime_hour_end = 5160,month = 1:12,day = as.character(1:30)
- eccc: system = as.character(1:5),year_start = 1993, leadtime_hour_end = 5136,month = 1:12,day = c('01')
- bom: system = as.character(2),year_start = 1993,leadtime_hour_end = 5208,month = 2:7,day = c('01'))

Structure of automated names

- 'seasonal_forecast'
- organization (e.g. dwd) with system (e.g. 21) –> dwd21
- variable (2m_temperature)
- years (e.g. 1996), if a range is supplied it covers min year to max year (e.g. 1996-2000)
- reference month (e.g. 11) this is the start point of the forecast, if a range is supplie it covers min month to max month (e.g. 1-12)
- leadtime (e.g 6), this is the hours since the reference month (usually starts at day 1 of the month), if range it covers max leadtime_hour to min leadtime_hour (e.g. 0-168 for a week forecast)
- area (e.g. 51-6.5-50-7.5), coordinates of map, follows the order: max latitude, min longitude, min latitude, max longtidue Example: season-forecast_dwd21_2m_temperature_1996_11_1_0-24_51-6.5-50-7.5 year: 1996, month: 11, leadtime: from 0 to 24 (from Nov 1 to Nov 2), area: 51, 6.5, 50, 7.5

How to run request and download indepently from another: The API usually needs some time (some minutes to some hours) to process the request and make the data available for download. So it can be convenient to handle data request and download seperately. If you set start_download = FALSE, then the function returns an object that allows to run the request later on. The object is an environment containing several variables and functions. To commence download, you can simply re-run the download function, this time supply the obtained object from the request to the variable 'request_env' If the data is ready for download, the download will start. If it still being processed or if it is still queued, the function would inform you about it and you can try it another time. You can also directly use the object obtained from the intial data-request to do the download manually. You can find an example down below.

Value

if start_download = TRUE, then nothing is returned. If start_download = FAlSE, then an environment object is returned, that can be used to download the requested data later (once it is ready for download).

Author(s)

Lars Caspersen

Examples

```
## Not run:

#download a simple dataset for a one week forecast
#takes one or two minutes
```

```
download_seasonal_forecast(year = c('1996'),
month = '11',
area = c(51, 6.5, 50, 7.5),
leadtime_hour = seq(0, 24*2, by = 6),
start_download = TRUE)

#use initial request and download independently from another
req <- download_seasonal_forecast(year = c('1996'),
month = '11',
area = c(51, 6.5, 50, 7.5),
leadtime_hour = seq(0, 24*2, by = 6),
start_download = FALSE)

#run download
download_seasonal_forecast(year = c('1996'),
month = '11',
area = c(51, 6.5, 50, 7.5),
leadtime_hour = seq(0, 24*2, by = 6),
start_download = FALSE,
request_env = req)

#you can also save the req_environment and run the download
#later in another r session
saveRDS(req, file = 'request.rds')
req <- readRDS('request.rds')


## End(Not run)
```

`ensemble_prediction_with_failure` *Make weighted mean prediction from scratch*

Description

Take model parameters, confidence scores for each set of parameters and calculates score-weighted mean prediction.

Usage

```
ensemble_prediction_with_failure(
  par_list,
  confidence,
  modelfn,
  temp,
  return_se = TRUE,
  n_fail = 5,
  max_weight = NULL,
  ...
)
```

Arguments

par_list	list of the parameters entries contain one read of the read with the function LarsChill::load_fitting_result(), each entry is a repetition of the same cultivar parameters are extracted from the 'xbest' entry within the individual list members I assumed ten model parameters with theta_star and Tc being fixed, but you can change that in the code after the line par <- x$xbest order of entries is identical with order in confidence (see next input)
confidence	gives the weights to the individual predictions numeric vector of same length as the par_list assumes that bigger is better
modelfn	function that takes one set of parameters and one entry of the seasonlist and returns a bloom date
temp	seasonlist containing hourly temperature data for the predictions generated with chillR::genSeasonList description
return_se	logical, decides if standard deviation of the predictions around the weighted mean should be returned as well
n_fail	numeric, decides the cut-off number of failure predictions of the weighted mean members so that the weighted mean also returns a failure in case n_fail = 5: if we have 4 or less failure predictions –> get ignored and the weighted mean is calculated based on remaining results if we have 5 or more failure predictions –> weighted mean is failure as well
max_weight	by default NULL, when number between 0 and 1 supplied, it expresses how much weight an individual prediction can get. Can prevent that one prediction dominates all the remaining ones because the confidence score may be inflated
...	further inputs for the modelfn argument

Value

when return_se = TRUE –> list with weighted mean, sd and individual model predictions when return_se = FALSE –> vector with weighted means

Author(s)

Lars Caspersen, <lars.caspersen@uni-bonn.de>

---

est_phen_gaps *Multiple linear shifts of bloom time window*

Description

Allows to shift properties of bloom time window from one location to another. Minimum requirement is at least one shared species between the locations.

Usage

```
est_phen_gaps(target_df, target_col, split_col)
```

Arguments

target_df	data.frame with phenological observations. Should contain a column called location and species
target_col	character, specifies for which column the shifts should be calculated
split_col	character, allows split into several groups if there are several phenological stages present. specifies the column which differentiates between the phenological stages

Value

data.frame with species, location, upper and lower limit in day of the year

Author(s)

Lars Caspersen

---

extract_seasonal_forecast

Opens downloaded seasonal forecast from the C3S database

Description

Opens a downloaded netcdf file (.nc) and brings it into a data.frame format. In case target latitude and longitude are provided, it extracts the values of the closest pixel. Otherwise it will extract all pixel values. The function was developed for precipitation but may also be compatible with other variables.

Usage

```
extract_seasonal_forecast(file, target_lat = NULL, target_lon = NULL)
```

Arguments

file	character, is the name of the file that needs to be extracted. In case the file is not in the working directory, the value needs to include either an absolute or relative path to the file.
target_lat	numeric, latitude of the location you want to extract. Be default set to NULL. If both target_lat and target_lon are NULL, then all the pixel values are extracted.
target_lon	numeric, longitude of the location you want to extract. Be default set to NULL. If both target_lat and target_lon are NULL, then all the pixel values are extracted.

Value

data.frame containing the columns: Year, Month, Day, Hour, temp, unit, model, latitude, longitude, target_lat, target_lon. Year, Month, Day, Hour indicate the date for the extracted value. temp: contains the extracted temperature values unit: indicates in what unit the temperatue is supplied (usually Kelvin) model: indicates the individual model that provided the output. Usually, the seasonal forecast contain an ensemble of models making the predictions. The underlying model is that generated the observation is shared among models, they are just different instances, as the forecast

involves randomness. latitude: original latitude of the pixel longtidue: original longitude of the pixel target_lat: latitude for the target point, that we wanted to extract target_lon: longitude of the taret point, that we wanted to extract

Author(s)

Lars Caspersen

Examples

```
## Not run:

#download a simple dataset for a one week forecast
#takes one or two minutes
download_seasonal_forecast(year = c('1996'),
month = '11',
area = c(51, 6.5, 50, 7.5),
leadtime_hour = seq(0, 24*7, by = 6),
start_download = TRUE)

fname <- 'season-forecast_dwd21_2m_temperature_1996_11_1_0-168_51-6.5-50-7.5.nc'

extract_seasonal_forecast(file = fname,
target_lat = 50.7,
target_lon = 7.1)


## End(Not run)
```

---

gen_combined_temp_response_plot

Draws temperature response plot from PhenoFlex parameters for several cultivars

---

Description

Returns a plot showing the chilling and forcing reaction of the PhenoFlex model parameters for certain temperatures.

Usage

```
gen_combined_temp_response_plot(
  par_df,
  weather_list,
  temps = seq(-5, 50, by = 0.1),
  legend.pos = "bottom",
 col_palette = c("#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0072B2", "#D55E00",
    "#CC79A7", "#999999"),
 par_names = c("yc", "zc", "s1", "Tu", "theta_star", "theta_c", "tau", "pie_c", "Tf",
    "Tc", "Tb", "slope"),
  chill_months = c(11:12, 1:2),
  heat_months = 1:3
)
```

Arguments

par_df	data.frame with at least the columns 'cultivar' and columns with the parameters name
weather_list	list with hourly weather observations, needed to create the histogram with the temperature frequency
temps	numeric, vector containing the temperatures for which the temperature responses should be calculated
legend.pos	character, by default 'bottom'. Specifies where the legend goes. Can take the same values as in ggplot2
col_palette	character vector, contains the hexcode of the colors used to draw the temperature responses for each cultivar
par_names	character, contains the names of the parameters, included in par_df. By default: c('yc', 'zc', 's1', 'Tu', 'theta_star', 'theta_c', 'tau', 'pie_c','Tf', 'Tc', 'Tb', 'slope')
chill_months	numeric vector, indicating for which months the frequency of observed temperature should be calculated for the chill temperature response. By default set to c(11:12,1:3)
heat_months	numeric vector, indicating for which months the frequency of observed temperature should be calculated for the heat temperature response. By default set to c(1:5)

Details

Assumes that the parameter data.frame has parameters in seperate columns and different cultivars. species etc. in rows. Runs on the twelve standard PhenoFlex parameters: yc, zc, s1, Tu, E0, E1, A0, A1, Tf, Tc, Tb, slope. In the actual calculation of chill and heat responses only the last 9 parameters are considered (and yc, zc and s1 are ignored).

Returns temperature response plot.#'

Value

ggplot of the modeled temperature response

Author(s)

Lars Caspersen

---

get_temp_response_df *Return temperature response*

Description

Applies constant temperatures to chill / heat model and returns temperature response.

Usage

```
get_temp_response_df(par, temp_values)
```

Arguments

par	traditional model parameters of PhenoFlex in the order yc, zc, s1, Tu, E0, E1, A0, A1, Tf, Tc, Tb, slope
temp_values	numeric, vector containing the temperatures for which the temperature responses should be calculated

Details

The output of the function is used in the temperature response plots, but it can be handy to create those manually, especially when comparing temperature responses of several cultivars or different set of parameters of same cultivar.

Value

data.frame with the columns "Temperature", "Chill_response" and "Heat_response"

Author(s)

Lars Caspersen

---

get_temp_response_plot

Draws temperature response plot from PhenoFlex parameters

Description

Returns a plot showing the chilling and forcing reaction of the PhenoFlex model parameters for certain temperatures.

Usage

```
get_temp_response_plot(
  par,
  temp_values,
  hourtemps = NULL,
  chill_months = c(11:12, 1:2),
  heat_months = 1:5,
  weather_freq_plot = "histogram"
)
```

Arguments

par	traditional model parameters of PhenoFlex in the order yc, zc, s1, Tu, E0, E1, A0, A1, Tf, Tc, Tb, slope
temp_values	numeric, vector containing the temperatures for which the temperature responses should be calculated
hourtemps	data.frame containing hourly temperature, by default set to NULL. If supplied, the columns 'Month' and 'Temp' should be present.
chill_months	numeric vector, indicating for which months the frequency of observed temperature should be calculated for the chill temperature response. By default set to c(11:12,1:3)

heat_months
: numeric vector, indicating for which months the frequency of observed temperature should be calculated for the heat temperature response. By default set to c(1:5)

weather_freq_plot
: character, only applicable when hourtemps is supplied. Decides how temperature observations should be represented in the temperature response plot. By default = 'histogram', which includes a histogram for the hourly temperature. Other option is 'gradient' which represents the frequency of observed temperature intervals in form of color gradient in the temperature response plot. Darker color indicates most frequent temperature intervals, bright color less frequent ones.

Details

Furthermore, it can highlight which temperatures where observed in the data set used to generate the model parameters. The function can handle the common PhenoFlex parameters (which include E0, E1, A0 and A1) and the "new" parameters (which include theta_star, theta_c, tau and pie_c instead of the earlier mentioned parameters).

Value

ggplot of the modeled temperature response

Author(s)

Lars Caspersen

---

get_thermal_window_phenology

Calculate bloom timewindow based on thermal risks

Description

Calculates risks for frost and heat and calculates upper and lower day of the year with viable flowering periods for each species and location.

Usage

```
get_thermal_window_phenology(
  weather_list_obs,
  weather_list_pred = NULL,
  observation_df,
  frost_threshold = 0,
  heat_threshold = 32,
  target_col_obs = "flowering_f50",
  run_mean_window = 10,
  padding = 0.05
)
```

Arguments

`weather_list_obs`
: list of daily weather observations, which need to contain at least the columns Date, Year, Month, Day, Tmin and Tmax. Names of list elements need to correspond to the names supplied in the `observation_df`. Based on this list, the risk thresholds are calculated tolerated by each species

`weather_list_pred`
: same structure as weather_list_obs. Based on this input the actual day ranges for each species and location are calculated. This allows for variable ranges with expected changes in weather. If NULL, then the weather_list_ob is used for that, too

`observation_df`
: data.frame containing the phenological observations. Need to contain the columns location (with the same names used as the named elements of weather_list_obs) and the phenological observations in Date format

`frost_threshold`
: numeric, by default 0. Decided the temperature in degree centigrade for which the lower temperature risk gets calculated.

`heat_threshold`
: numeric, by default 32. Decides the temperature in degree centigrade for which the higher temperature risk gets calculated.

`target_col_obs`
: character, by default 'flowering_f50'. Specifies the column name of observation_df where the phenological records are stored.

`run_mean_window`
: numeric, by default 10. Decides the window length of the running mean, which is used to smooth the calculated risks.

`padding`
: numeric, by default 0.05. Specifies how much extra risk tolerance is added to the calculated risk thresholds (between 0 and 1)

Details

there are several assumptions in the calculation of time windows. Lower day of the year cannot be lower than 1. Upper end is assumed to be before the maximum heat risk. If calculated heat tolerance of a species exceeds the maximum risk of a location, then the upper end of the bloom window is assumed to be the peak of heat risk.

Value

data.frame containing the species, location, upper day of the year and lower day of the year for the bloom time window

Author(s)

Lars Caspersen

load_fitting_result *Load fitted PhenoFlex model*

Description

Allows to read fitted PhenoFlex models, which were saved using save_fitting_list

Usage

```
load_fitting_result(path, prefix)
```

Arguments

path	character, specifying the location (relative to the working directory)
prefix	character, added in front of the standard file name

Details

This function reads the saved PhenoFlex model to a list.

Value

list with the fitted objects. Elements are named after the files (without numbering and prefix)

Author(s)

Lars Caspersen

modified_ComprehensivePrecipitationGenerator *This is a modified version of the ComprehensivePrecipitationGenerator*

Description

The difference to the original version is, that the function crashes if the spline interpolation of mean daily rainfall by month yields values close to zero.

Function to incorporate the temperature and precipitation 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 and mean daily rainfall (mm) can be specified to allow generation of temperature change scenarios.

Usage

```
modified_ComprehensivePrecipitationGenerator(
  station = c("T0001", "T0010", "T0099"),
  prec_all,
  mean_climate_prec = NULL,
  year_max = 1990,
  year_min = 1961,
  leap = TRUE,
  nmonth = 12,
  cpf = NULL,
  verbose = TRUE,
  p = 1,
  type = "none",
  lag.max = NULL,
  ic = "AIC",
  activateVARselect = FALSE,
  exogen = NULL,
  exogen_sim = NULL,
  is_exogen_gaussian = FALSE,
  year_max_sim = year_max,
  year_min_sim = year_min,
  mean_climate_prec_sim = NULL,
  onlygeneration = FALSE,
  varmodel = NULL,
  type_quantile = 3,
  qnull = NULL,
  valmin = 0.5,
  step = 0,
  n_GPCA_iteration = 0,
  n_GPCA_iteration_residuals = n_GPCA_iteration,
  sample = NULL,
  extremes = TRUE,
  exogen_all = NULL,
  exogen_all_col = station,
  no_spline = FALSE,
  nscenario = 1,
  seed = NULL,
  noise = NULL
)

temperature_generation_rmawgen_prec(
  weather,
  years,
  sim_years,
 temperature_scenario = data.frame(Tmin = rep(0, 12), Tmax = rep(0, 12), Prec = 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

station	station names
prec_all	matrix with precipitation in mm for all stations. one column per station
mean_climate_prec	by default NULL
year_max	numeric, starting year
year_min	numeric, end year
leap	decides if leap years are included in simulated data
nmonth	how many months are to be expected in input data
cpf	don't know
verbose	if message are printed while running function
p	don't know
type	dont't know
lag.max	lag considered when simulating temperature and or precipitation
ic	don't know
activateVARselect	don't know
exogen	if there are exogenous variables to be considered when generating weather
exogen_sim	don't know
is_exogen_gaussian	don't know
year_max_sim	don't know
year_min_sim	don't know
mean_climate_prec_sim	don't know
onlygeneration	too lazy to document
varmodel	too lazy to document
type_quantile	too lazy to document
qnull	too lazy to document
valmin	too lazy to document
step	too lazy to document
n_GPCA_iteration	too lazy to document
n_GPCA_iteration_residuals	too lazy to document
sample	too lazy to document
extremes	too lazy to document
exogen_all	too lazy to document
exogen_all_col	too lazy to document
no_spline	too lazy to document

`nscenario` too lazy to document

`seed` integer specifying the random seed for the weather generation.

`noise` too lazy to document

`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.

`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

This function is called by the wrapper function `temperature_generation_rmawgen_prec`. The rest of the documentation is copied from the original function.

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","T "Prec"). If temperature_scenario is a list, the output list contains simulated temperature records for all scenarios.

list of data.frames containing the simulated weather, with columns c("YEARMODA", "DATE","Year","Month","Day","T "Prec"). If temperature_scenario is a list, the output list contains simulated temperature records for all scenarios.

Author(s)

Lars Caspersen, Eike Luedeling

Lars Caspersen

`mva_bias_correction_forecast`

Perform mean and variance adjustment (MVA) bias correction

Description

Calculates mean biases and standard deviation for forecast data and local observed temperature data. Performs bias correction on monthly means, then uses the ratio of corrected and uncorrected means to scale the daily and sub-daily observations. Follows procedure outlined by: Manazanas et al. (2019) and Leung et al. (1999). Returns a data.frame with corrected and original forecast data.

Usage

```
mva_bias_correction_forecast(observed, predicted, unit_observed = "C")
```

Arguments

`observed`	data.frame, contains observed temperature. Is usually the same format as when downloaded via chillR package. Should contain columns: Year, Month, Day, and a column with temperature. Valid options are: Tmean, Temp, or combination of Tmin and Tmax.
`predicted`	data.frame containing the forecast data. Usually the same format as when running the extract_seaonal_forecast() function. Should contain columns: Year, Month, Day, Temp, unit
`unit_observed`	character, by default "C". Indicates the unit of the input temperature. Bias correction is done in Kelvin. To indicate Fahrenheit, use "F". To indicate Kelvin use "K"

Details

Bias correction does three steps for each month of the year:

- Calculate monthly means and sd for observed and forecasted data
- Correct forecasted monthly mean with the formula: (mean_fcst_m_t - mean_fcst) x (sd_obs / sd_fcst) + mean_obs
- Scale (sub) daily observation using share of corrected and uncorrected forecast mean

mean_fcst_m_t = monthly mean of forecast member m, at year t mean_fcst = monthly mean of all forecast members at all years sd_obs = standard deviation of observed monthly means across all years df_fcst = standard deviation of monthly means across all members m, across all years mean_obs = monthly mean of observed temperature, across all years

Important note: as the function corrects using a factor, it is sensitive to sub-zero values. The function carries the mean correction out in Kelvin behin the scenes.

Value

data.frame, same format as "predicted", with additional column indicating correction factor and corrected (sub) daily temperature

Author(s)

Lars Caspersen

Examples

```
## Not run:

#download a simple dataset for a one week forecast
#takes one or two minutes

req <- download_seasonal_forecast(year = 1993:2000,
month = '11',
area = c(51, 6.5, 50, 7.5),
leadtime_hour = seq(0, 24*30, by = 6),
start_download = FALSE)

#run download
req <- download_seasonal_forecast(year = 1993:2000,
month = '11',
area = c(51, 6.5, 50, 7.5),
leadtime_hour = seq(0, 24*30, by = 6),
start_download = FALSE,
request_env = req)

#extract values
fname <- 'season-forecast_dwd21_2m_temperature_1996_11_1_0-168_51-6.5-50-7.5.nc'

extract_seasonal_forecast(file = fname,
target_lat = 50.7,
target_lon = 7.1)

#download local observation
long <- 7.0871843
lat <- 50.7341602
weather_dwd <- chillR::handle_dwd(action = 'list_stations',
                                  location = c(long, lat),
                                  time_interval = c(19800101, 20251231))

data <- chillR::handle_dwd(action = "download_weather",
                           location = weather_dwd[1:3, "Station_ID"],
                           time_interval = c(19800101, 20241231),
                           stations_to_choose_from = 25,
                           station_list = weather_dwd,
```

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

  data_clean <- chillR::handle_dwd(data)

  observed <- data_clean$`Köln/Bonn`

  predicted_bias_corrected <- mva_bias_correction_forecast(observed, predicted)


  ## End(Not run)
```

`phenoflex_parnames_new` *PhenoFlex Parameter Names (New Version)*

Description

A character vector containing parameter names for the new version of Phenoflex.

Usage

```
data(phenoflex_parnames_new)
```

Format

A character vector with parameter names.

Examples

```
data(phenoflex_parnames_new)
head(phenoflex_parnames_new)
```

`phenoflex_parnames_old` *PhenoFlex Parameter Names (Old Version)*

Description

A character vector containing parameter names for the old version of Phenoflex.

Usage

```
data(phenoflex_parnames_old)
```

Format

A character vector with parameter names.

Examples

```
data(phenoflex_parnames_old)
head(phenoflex_parnames_old)
```

`pheno_ensemble_prediction` *Get weighted average prediction of several PhenoFlex models*

Description

Outdated

Usage

```
pheno_ensemble_prediction(
  par_list,
  confidence,
  modelfn,
  temp,
  return_se = TRUE,
  ...
)
```

Arguments

`par_list`	the output of the phenology fitting done using MEIGO package
`confidence`	numeric vector, containing the weights assign to the predictions. Assumes that higher value is more confidence. Values get re-scaled by dividing by the sum of individual confidence scores.
`modelfn`	function that takes one set of parameters and one entry of temp as input and returns bloomdate
`temp`	SeasonList for the prediction, contains the hourly weather data of the weather stations
`return_se`	boolean, by default set TRUE. Decides if sd of prediction and individual ensemble members predictions are returned as well
`...`	further inputs for modelfn

Details

This function takes the fitted models and temperature data and returns a weighted average prediction. Weights are assigned using the confidence argument. The function assumes that theta_star and Tc are fixed. The

Value

By default returns a list with three components: predicted, sd and individual_pred. Predicted contains the weighted predictions of the ensemble members. sd contains the unweighted standard deviation of the members predictions. Individual_pred contains the ensemble members individual predictions

Author(s)

Lars Caspersen

`return_predicted_days` *Returns predicted bloomdays for a list of temperature time series and model parameters*

Description

This is a convenience function which allows to get the predicted bloom dates for a set of model parameters and temperature observations. #'

Usage

```
return_predicted_days(par, modelfn, SeasonList, na_penalty = 365, ...)
```

Arguments

`par`	traditional model parameters of PhenoFlex in the order yc, zc, s1, Tu, E0, E1, A0, A1, Tf, Tc, Tb, slope.
`modelfn`	function used within the evaluation function to calculate the actual bloomday, often we use the 'custom_GDH_wrapper' function for that
`SeasonList`	list of hourly temperatures for the individual phenological seasons. Each element should contain a data.frame with the columns "Temp" (for the hourly temperature) and "JDay" for the corresponding Julian day. Is usually generated using genSeasonList
`na_penalty`	numeric, value which is used when the model fails to generate a prediction for the bloom date. By default 365
`...`	further arguments for modelfn

Value

numeric vector with the predicted bloom dates

Author(s)

Lars Caspersen

save_fitting_list *Save fitted PhenoFlex model*

Description

Allows to save fitted PhenoFlex models. They can be loaded again using load_fitting_result

Usage

```
save_fitting_list(fit_list, path, prefix)
```

Arguments

fit_list	list with the fitted PhenoFlex models. Assumes that it is a list with at least one model. Elements should be named, because these will be used when creating the file names.
path	character, specifying the location (relative to the working directory)
prefix	character, added in front of the standard file name

Details

This function saves the fitted model with all the additional information provided by MEIGOR function and saves it to .txt files.

Value

list with the fitted objects. Elements are named after the files (without numbering and prefix)

Author(s)

Lars Caspersen

solve_nle *Function used to calculate E0, E1, A0 and A1 of Phenoflex*

Description

This function is used with the nlseq packages to to convert the parameters theta_star, theta_c, tau and pie_c to E0, E1, A0 and A1. Parameters E0 and E1 need to be numerically approximated. The function returns the error of the approximation for the given set of parameters.

Usage

```
solve_nle(x, params)
```

Arguments

x	numeric vector, containing values of E0 and E1
params	numeric vector, containing values of parameters theta_star, theta_c, tau and pie_c the 'custom_GDH_wrapper' function for that

Value

error, for the current set of E0 and E1, which gets minimized during the approximation

Author(s)

Lars Caspersen

`weighted_mean_with_fail` *Process data.frame of predictions to weighted mean prediction*

Description

Helps to process already made predictions to weighted mean predictions. Has a routine how to handle predictions of bloom failure. The function makes strong assumptions on the data frame of the predictions and the data frame containing the weights (confidence)

Usage

```
weighted_mean_with_fail(
  predicted,
  confidence,
  return_se = TRUE,
  n_fail = 5,
  max_weight = NULL,
  .progress = FALSE
)
```

Arguments

`predicted`	data.frame with individual predictions, need to contain column called id role of id column is to match predictions with the entries of the confidence input so each cultivar would get their own id, can be the name, or something else assume wide format –> so one column for each prediction that means for one particular year we have ten columns with the ten predictions for that year (assuming you have ten repetitions) function assumes that the names of the columns containing the predictions will be also present in the confidence input so if the predictions are stored in columns R1 to R10, these column names need to be present in confidence, too.
`confidence`	data.frame, needs to contain same columns as predicted (through predicted can have many more additional columns) assume wide format, so 10 columns when ten repetitions, plus id assumes that larger values mean more confidence id entries need to match with id entries of the first input ('predicted')
`return_se`	logical, decides if standard deviation of the predictions around the weighted mean should be returned as well
`n_fail`	numeric, decides the cut-off number of failure predictions of the weighted mean members so that the weighted mean also returns a failure in case n_fail = 5: if we have 4 or less failure predictions –> get ignored and the weighted mean is calculated based on remaining results if we have 5 or more failure predictions –> weighted mean is failure as well

max_weight	by default NULL, when number between 0 and 1 supplied, it expresses how much weight an individual prediction can get. Can prevent that one prediction dominates all the remaining ones because the confidence score may be inflated
.progress	logical, if set TRUE than process par appears indicating the remaining time until the function completes

Value

same as 'predicted' but with additional column containing weighted mean and standard deviation

Author(s)

Lars Caspersen, <lars.caspersen@uni-bonn.de>

---

wrapper_prepare_shift_plot

Prepare a the data.frame to make a shift plot

Description

Takes a data.frame as an input, usually the result of some kind of climate change impact projection. Prepares data, so that I can make easily shift plots as in Caspersen et al. (in review), or the analysis of apple phenology at lake Constance.

Usage

```
wrapper_prepare_shift_plot(
  shift_df,
  group_col,
  cut_share = c(0, 0.2, 0.4, 0.6, 0.8, 1)
)
```

Arguments

shift_df	data.frame containing the summarized climate change impact projections and the baseline values. Assumes that it is already summarized (e.g. by median). Assumes that the difference of future and baseline is summarized in the column "shift". Assumes baseline is summarized in column "baseline".
group_col	character vector, containing the column names to group the shift data. Usually this includes the climate scenario,the scenario year and cultivar name. All columns need to be present in shift_df.
cut_share	numeric vector, segments of the shift plot. Should be between 0 and 1. For each interval, the function will calculate the percentage of shifts within or exceeding the interval#'

Details

Makes relative strong assumptions on the structure of the input. Make sure to format it accordingly.

Value

data.frame, with additional columns for the intervals and the agreement among the climate projections regarding the intensity of the shift

Author(s)

Lars Caspersen

Index