Package 'openQUARREL'

August 25, 2025

Title A Meta-Package Integrating Several Hydrological Model Packages in R

Version 1.1.0

Description Provides a unified interface to several hydrological model packages described in Astagneau et al. (2021), https://doi.org/10.5194/hess-25-3937-2021, Includes tools for model calibration, ensemble simulation, and performance evaluation.

License GPL-3

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Imports
airGR, DEoptim, dplyr, gdata, grDevices, graphics, hydromad, hydroPSO, lhs, magrittr, methods, purrr, randtoolbox, readr, Rmalschains, stats, stringr, tidyr, topmodel, TUWmodel, zoo

Depends R (>= 4.1.0)

Suggests hydroGOF,
knitr,
rmarkdown

Remotes JosephGuillaume/hydromad@oldstable, hzambran/hydroPSO

VignetteBuilder knitr

URL https://mw-schirmer.github.io/openQUARREL/

R topics documented:

`calc_crit` *Calculate Performance Criterion*

Description

Computes a performance criterion for comparing simulated and observed runoff. If the criterion is `"KGEtang"`, it uses the `KGEtang` function. Otherwise, it delegates to `calc_hydroGOF()` for other supported criteria.

Usage

```
calc_crit(error_crit, Qsim, Qobs)
```

Arguments

`error_crit`	A string specifying the error criterion to compute. Supported values include `"KGEtang"` and any criterion accepted by `calc_hydroGOF()`.
`Qsim`	A numeric vector of simulated runoff values.
`Qobs`	A numeric vector of observed runoff values.

Value

A numeric value representing the selected performance criterion.

See Also

`KGEtang`, `calc_hydroGOF`

Examples

```
Qsim <- c(1, 2, 3, 4, 5)
Qobs <- c(1.1, 2.1, 2.9, 4.2, 5.1)
calc_crit("KGEtang", Qsim, Qobs)
```

`calc_hydroGOF` *Wrapper function around* `hydroGOF` *functions*

Description

Calculates Goodness-of-Fit functions for two runoff series

Usage

```
calc_hydroGOF(GOF_fun, Qsim, Qobs, na.rm = TRUE)
```

Arguments

`GOF_fun`	a function, or (todo consider only a functional) a string with function name, of the format `GOF_fun(Qsim, Qobs, na.rm = "TRUE")`, typically from the `hydroGOF` package
`Qsim`	vector, matrix, data.frame etc of simulated runoff values
`Qobs`	vector, matrix, data.frame etc of observed runoff values
`na.rm`	a logical value indicating if NA should be removed

Value

transformed runoff in same format as input

Note

This function requires the `hydroGOF` package to be installed. Not imported as this package depends on deprecated packages as sp etc.

See Also

`hydroGOF`

Examples

```
# hydroGOF must be loaded
library(hydroGOF)
calc_hydroGOF(KGE, 1:10, seq(0, 9))
# this is NA
calc_hydroGOF("KGE", 1:10, rep(0, 10))
# this is also NA
calc_hydroGOF(KGE, 1:10, as.numeric(rep(NA, 10)))
```

`calc_subseasonal_validation_results` *Calculate Subseasonal Validation Metrics*

Description

Computes validation metrics for specified subseasonal periods within a hydrological dataset. For each named period in `val_subseason`, the function subsets the data using the provided indices and calculates performance metrics using `calc_validation_results`.

Usage

```
calc_subseasonal_validation_results(
  val_subseason,
  dates,
  ind,
  period_name,
  col_name = "period",
  Qsim,
  Qobs,
  val_crit_transfo = "KGE__none"
)
```

Arguments

`val_subseason`	A named list where each element is a character vector of two-digit month codes (e.g., `"06"`, `"07"`) defining the months for each subseasonal period.
`dates`	A vector of dates (e.g., of class `Date` or `POSIXct`) corresponding to the time series.
`ind`	A vector of indices used to subset the hydrological data.
`period_name`	A string indicating the name of the period (e.g., `"calibration"` or `"validation"`), which will be added as a column in the output.
`col_name`	A string specifying the name of the additional column to be added to the output (default is `"period"`).
`Qsim`	A numeric vector of simulated runoff values.
`Qobs`	A numeric vector of observed runoff values.
`val_crit_transfo`	A character vector specifying validation criteria and runoff transformations, separated by `"__"`. Optionally, a third part can specify a lambda parameter. Supported criteria are those from the `hydroGOF` package and must be compatible with `calc_hydroGOF`. Supported transformations are described in `transfo_q`.

Value

A data frame similar to the output of `validate_model`, but with two additional columns: one for the subseasonal period (e.g., `"spring"`, `"summer"`) and one for the period name (e.g., `"calibration"`), as specified by `col_name` and `period_name`.

See Also

calc_validation_results, validate_model, calc_hydroGOF, transfo_q

Examples

```
perf_cal <- calc_subseasonal_validation_results(
  val_subseason = list(
    spring = c("02", "03", "04", "05"),
    summer = c("06", "07", "08", "09")
  ),
  dates = hydro_data$BasinObs$DatesR,
  ind = split_indices$ind_cal,
  period_name = "calibration",
  col_name = "period",
  Qsim = simulation_results$Qsim,
  Qobs = Qobs,
  val_crit_transfo = c(
    "KGE__none", "NSE__none", "VE__none", "pbias__none",
    "KGE__inv", "NSE__inv",
    "KGE__sqrt", "NSE__sqrt"
  )
)
```

---

calc_validation_results

Validate Hydrological Model Performance

Description

Calculates validation metrics for simulated versus observed runoff using various transformation types.

Usage

```
calc_validation_results(
  ind,
  period_name,
  col_name = "period",
  Qsim,
  Qobs,
  val_crit_transfo = "KGE__none"
)
```

Arguments

Qsim	A numeric vector of simulated runoff values.
Qobs	A numeric vector of observed runoff values.
val_crit_transfo	A character vector specifying validation criteria and runoff transformations, separated by "__". Optionally, a third part can specify a lambda parameter. Supported criteria are those from the hydroGOF package and must be compatible with calc_hydroGOF. Supported transformations are described in transfo_q.

Details

The function splits each entry in `val_crit_transfo` into components: validation criterion, transformation type, and optionally a lambda value. It applies the transformation to both `Qsim` and `Qobs`, computes the specified metric, and returns the results in a tidy format.

Value

A long-format data frame with columns:

- `crit` – the validation criterion used (e.g., "KGE", "NSE").
- `transfo` – the runoff transformation applied (e.g., "log", "inv", "none").
- `lambda` – the lambda parameter used for transformation, if applicable.
- `value` – the resulting validation metric value.

Examples

```
validate_model(
  Qsim = 1:10,
  Qobs = seq(2, 11),
  val_crit_transfo = c("KGE__log", "NSE__inv", "VE__none", "pbias__none")
)
```

`calibrate_model` *Calibrate a Hydrological Model*

Description

Performs calibration of a hydrological model using a specified optimization algorithm. Supports models from the **airGR**, **TUWmodel**, **hydromad**, and **topmodel** packages, with optional snow module integration. The function supports both native calibration routines (e.g., `Calibration_Michel` for **airGR**) and general-purpose optimizers (e.g., `DEoptim`, `hydroPSO`, `malschains`).

Usage

```
calibrate_model(
  hydro_data,
  split_indices,
  model,
  input,
  snow_module = NULL,
  snow_input = NULL,
  snow_parameters = NULL,
  error_crit_transfo = "KGE__none",
  cal_maximize = TRUE,
  cal_fn = "DEoptim",
  do_transfo_param = FALSE,
  cal_par = default_cal_par
)
```

Arguments

hydro_data	A list or data frame containing observed runoff and meteorological data, typically loaded using load_meteo_data.
split_indices	A list of index vectors (e.g., from split_data_set) indicating warm-up and calibration periods.
model	A string specifying the hydrological model to calibrate. For a complete list see table in vignette("model_overview").
input	A list of model input data, typically created using create_input.
snow_module	Optional. A string specifying the snow module (currently "CemaNeige" and "TUWsnow").
snow_input	Optional. Input data for the snow module.
snow_parameters	Optional. A vector of fixed snow parameters. If NULL, snow parameters are assumed to be part of the calibration.
error_crit_transfo	A string combining the error criterion and runoff transformation, separated by "__" (e.g., "KGE__none"). Optionally, a third value (e.g., lambda) can be included for transformations like Box-Cox.
cal_maximize	Logical. If TRUE, the calibration maximizes the objective function.
cal_fn	A string specifying the calibration function. Supported options include "Calibration_Michel" (for **airGR** models), "DEoptim", "hydroPSO", "malschains", and other supported optimizers. For a complete list see table in vignette("calibration_methods_overview").
do_transfo_param	Logical. If TRUE, parameters are transformed to a unit hypercube before calibration.
cal_par	A list of calibration settings specific to the chosen calibration function. Defaults to default_cal_par, but can be customized by the user.

Value

A list containing:

- model_param: Calibrated model parameters.
- error_crit_transfo: The error criterion and transformation used.
- error_crit_val: The final value of the error criterion.
- cal_fn: The calibration function used.
- do_transfo_param: Whether parameter transformation was applied.
- duration: Duration of the calibration process.
- cal_par: Calibration settings used.
- more_info: Additional model- or method-specific output.

Note

- Calibration_Michel is only available for **airGR** models.
- If Calibration_Michel is used, parameters are assumed to be transformed.
- The function supports power and Box-Cox runoff transformations, with lambda optionally specified.
- Future improvements may simplify access to cal_par for end users.

See Also

call_cal_fn, optim_fn, create_input, load_meteo_data, split_data_set, transfo_q

Examples

```
## Not run:
calibration_results <- calibrate_model(
  hydro_data = hydro_data,
  split_indices = split_data_set(...),
  model = "GR4J",
  input = create_input(...),
  error_crit_transfo = "KGE__none",
  cal_maximize = TRUE,
  cal_fn = "DEoptim",
  do_transfo_param = TRUE,
  cal_par = default_cal_par
)

## End(Not run)
```

---

call_cal_fn *Call Calibration Function for Hydrological Model*

Description

Executes a specified optimization algorithm to calibrate a hydrological model. Supports multiple calibration methods and Monte Carlo sampling strategies.

Usage

```
call_cal_fn(
  cal_fn,
  hydro_data,
  split_indices,
  model,
  input,
  snow_module = NULL,
  snow_input = NULL,
  snow_parameters = NULL,
  error_crit,
  cal_maximize,
  cal_q_transfo,
  lambda,
  do_transfo_param,
  cal_par = default_cal_par
)
```

Arguments

cal_fn	A string specifying the calibration function or method to use. Supported options include "DEoptim", "hydroPSO", and "malschains". Monte Carlo variants can be specified using the format "method__sampling__nruns". For a complete list see table in vignette("calibration_methods_overview").

hydro_data	A list or data frame containing observed runoff data, typically loaded using load_meteo_data.
split_indices	A list of index vectors (e.g., from split_data_set) indicating warm-up and calibration periods.
model	A string specifying the hydrological model to calibrate. For a complete list see table in vignette("model_overview").
input	A list of model input data, typically created using create_input.
snow_module	Optional. A string specifying the snow module used (currently "CemaNeige" and "TUWsnow").
snow_input	Optional. Input data for the snow module.
snow_parameters	Optional. Initial or fixed parameters for the snow module.
error_crit	A string naming the error criterion function (e.g., "KGE"). Must be compatible with calc_hydroGOF or from the **hydroGOF** package.
cal_maximize	Logical. If TRUE, the calibration maximizes the objective function.
cal_q_transfo	A string indicating how runoff should be transformed (see transfo_q).
lambda	Optional. A numeric value or vector used for regularization or multi-objective weighting.
do_transfo_param	Logical. If TRUE, parameters are transformed to a unit hypercube before calibration.
cal_par	A list of calibration settings specific to the chosen calibration function. Defaults to default_cal_par, but can be customized by the user.

Value

A list or object containing the results of the calibration, including optimized parameters and performance metrics.

Note

This function requires the hydroPSO package to be installed. Not imported as this package depends on deprecated packages as sp etc.

Examples

```
## Not run:
cal_output <- call_cal_fn(
  cal_fn = "DEoptim",
  hydro_data = hydro_data,
  split_indices = split_data_set(...),
  model = "GR4J",
  input = create_input(...),
  error_crit = "KGE",
  cal_maximize = TRUE,
  cal_q_transfo = "none",
  do_transfo_param = TRUE,
  cal_par = default_cal_par
)

## End(Not run)
```

create_hydromad_model *Create a hydromad Model Object*

Description

Constructs a hydromad model using a specified soil moisture accounting (SMA) routine and a set of default calibration parameters.

Usage

```
create_hydromad_model(sma, cal_par, routing = "expuh")
```

Arguments

sma	A string specifying the soil moisture accounting model (e.g., "cmd", "gr4j", "sacramento").
cal_par	A list of calibration parameters, typically created using set_hydromad_par. Must contain lower and upper bounds for each parameter.
routing	A string specifying the routing model (default is "expuh").

Details

The function initializes a hydromad model with no data and updates it with the parameter bounds provided in cal_par. This is useful for preparing a model structure before calibration.

Value

A hydromad model object with parameter ranges set.

Examples

```
create_hydromad_model("cmd", default_cal_par[["cmd"]])
```

create_input *Create Model Input Structure*

Description

Creates an input structure tailored to the specified hydrological model. The function supports models from the **airGR**, **TUWmodel**, **hydromad**, and **topmodel** packages. The structure and content of the input depend on the model's requirements.

Usage

```
create_input(model, BasinObs, BasinInfo)
```

Arguments

model	A string specifying the hydrological model (e.g., "GR4J", "TUW", "topmodel"). For a complete list see table in vignette("model_overview").
BasinObs	A data frame containing time series of meteorological and hydrological data, typically created using load_meteo_data.
BasinInfo	A list containing spatial and catchment-specific information such as HypsoData, delay, or topidx, depending on the model.

Details

- For airGR models, the function returns an object created by airGR::CreateInputsModel().
- For TUW, TUWsnow, snowsnow, and hydromad models, a renamed data frame is returned.
- For topmodel, a list is returned including additional elements like delay and topidx.

Value

A model-specific input object, either a data frame, list, or S3 class, depending on the model type.

See Also

load_meteo_data, get_family, airGR::CreateInputsModel()

Examples

```
## Not run:
BasinObs <- load_meteo_data("D:/input/airGR/HSU_2044.rds")
BasinInfo <- list(HypsoData = c(500, 600, 700), delay = 2, topidx = runif(10))
input <- create_input("GR4J", BasinObs, BasinInfo)

## End(Not run)
```

---

default_cal_par *Default calibration parameters*

Description

Default calibration parameters

Usage

```
default_cal_par
```

Format

An object of class list of length 17.

find_monthly_indices *Find Monthly Indices in a Date Vector*

Description

Returns the indices of dates that fall within specified months. Useful for subsetting time series data by month.

Usage

```
find_monthly_indices(date, months, ind = seq_along(date))
```

Arguments

date	A vector of dates (e.g., Date, POSIXt).
months	A character vector of two-digit month strings (e.g., c("02", "03")).
ind	An optional vector of indices used to subset the input date vector before filtering. Defaults to all indices in the date vector.

Value

An integer vector of indices corresponding to

get_family *Get the Family or Package of a Hydrological Model*

Description

Performs a reverse lookup to identify the package or model family associated with a given hydrological model name.

Usage

```
get_family(model_str)
```

Arguments

model_str	A string specifying the name of a hydrological model (e.g., "GR4J").

Value

A string indicating the model's package or family name. Returns "none" if the model is not found.

Examples

```
get_family("GR4J")
get_family("UnknownModel")
```

hydro_family *Hydrofamily or Package*

Description

Hydrofamily or Package

Usage

```
hydro_family
```

Format

An object of class `data.frame` with 16 rows and 2 columns.

input_data *Input data for the hydrological models in vignette*

Description

Precipitation, temperature, potential evaporation and observed discharge

Usage

```
input_data
```

Format

`input_data`**:**

A data frame with 148,530 rows and 7 columns:

HSU_ID The ID of the catchment used by the federal office for the environment (FOEN)

DatesR Dates

P Precipitation in mm/d

T Temperature in deg Celsius

E Potential evaporation in mm/d

Qm3s observed discharge in m3/s

Qmm observed discharge in mm/d

Source

https://www.meteoswiss.admin.ch/climate/the-climate-of-switzerland/spatial-climate-analyses.html

https://www.bafu.admin.ch/bafu/en/home/topics/water/data-and-maps/water-monitoring-data/hydrological-data-service-for-watercourses-and-lakes.html

KGEtang *Kling-Gupta Efficiency (KGE") after Tang et al. (2021)*

Description

Computes the modified Kling-Gupta Efficiency (KGE) as proposed by Tang et al. (2021), https://doi.org/10.1175/jcli-d-21-0067.1, Equation (5)

Usage

```
KGEtang(Qsim, Qobs, single_output = TRUE, ...)
```

Arguments

Qsim	A numeric vector of simulated runoff values.
Qobs	A numeric vector of observed runoff values.
single_output	A boolean if a single combined or all components should be output

load_meteo_data *Load Meteorological Data*

Description

Loads a data frame stored in an .rds file containing meteorological data formatted similar to the airGR model family. The input must include specific columns as described below.

Usage

```
load_meteo_data(file, tzone = "UTC")
```

Arguments

file	A string specifying the path to the .rds file containing the meteorological data.
tzone	A string specifying the time zone to assign to the DatesR column. If "UTC" (default), a Date vector is not converted to another time zone.

Details

The input data frame must contain the following columns:

- DatesR: Dates in Date or POSIXt format
- P: Average precipitation [mm/day]
- T: Catchment average air temperature [°C]
- E: Catchment average potential evapotranspiration [mm/day]
- Qmm: Outlet discharge [mm/day]

For more information on the required format, see the airGR documentation: https://hydrogr.github.io/airGR/page_1_get_started.html

Value

A data frame named `BasinObs` containing the loaded and time-adjusted data.

Examples

```
## Not run:
load_meteo_data("D:/input/airGR/HSU_2044.rds")

## End(Not run)
```

`merge_snow_runoff_sim` *Merge Snow and Runoff Simulation Results*

Description

Combines the output of a snow module simulation with the results of a hydrological runoff simulation. Adds snow-related variables and snow model metadata to the runoff simulation results.

Usage

```
merge_snow_runoff_sim(simulation_results, snow_module_results)
```

Arguments

`simulation_results`
: A list returned by `simulate_model`, containing simulated runoff and related metadata.

`snow_module_results`
: A list returned by `simulate_snow`, containing snow-related outputs such as SWE, melt, and precipitation components.

Value

A list identical to `simulation_results`, but with additional elements:

- `SWE`: Snow water equivalent.
- `psolid`: Solid precipitation.
- `pliquid`: Liquid precipitation.
- `melt`: Meltwater.
- `surface_water_runoff`: Combined snow runoff.
- `more_info$snow_module`: Snow model output metadata.

See Also

`simulate_model`, `simulate_snow`

Examples

```
## Not run:
merged_results <- merge_snow_runoff_sim(simulation_results, snow_module_results)

## End(Not run)
```

norm_minmax *Min-Max Normalization and Re-Transformation*

Description

Scales numeric data to the [0, 1] range using min-max normalization, or re-transforms normalized data back to its original scale. The behavior depends on the specified direction.

Usage

```
norm_minmax(x, min, max, direction = "RT")
```

Arguments

x	A numeric vector, matrix, or array to be scaled or re-transformed.
min	The minimum value used for scaling. Required for both directions, but typically set to `min(x)` when `direction = "RT"`.
max	The maximum value used for scaling. Required for both directions, but typically set to `max(x)` when `direction = "RT"`.
direction	A character string indicating the direction of transformation: `"RT"` (real to transformed) for normalization, or `"TR"` (transformed to real) for re-scaling. Default is `"RT"`.

Value

A numeric object of the same shape as `x`, either normalized or

optim_fn *Objective Function for Hydrological Model Calibration*

Description

This function is used as the objective function during model calibration. It simulates a hydrological model over the warm-up and calibration periods and evaluates an error criterion (e.g., KGE) on the calibration period only.

Usage

```
optim_fn(
  ParamOptim,
  hydro_data,
  split_indices,
  model,
  input,
  snow_module = NULL,
  snow_input,
  snow_parameters = NULL,
  error_crit,
  cal_maximize,
```

```
  cal_q_transfo,
  lambda,
  do_transfo_param,
  airGR_RunOptions = NULL,
  airGR_RunOptions_snow_module = NULL
)
```

Arguments

ParamOptim	A numeric vector of model parameters to be optimized.
hydro_data	A list or data frame containing observed runoff, typically loaded using load_meteo_data.
split_indices	A list of index vectors indicating warm-up and calibration periods, usually from split_data_set.
model	A string specifying the hydrological model. For a complete list see table in vignette("model_overview").
input	A list of model input data, typically created using create_input.
snow_module	Optional. A string specifying the snow module (currently "CemaNeige" and "TUWsnow").
snow_input	Optional. Input data for the snow module.
snow_parameters	Optional. A vector of fixed snow parameters. If NULL, snow parameters are assumed to be part of ParamOptim.
error_crit	A string naming the error criterion function (e.g., "KGE"). Must be compatible with calc_hydroGOF or from the **hydroGOF** package.
cal_maximize	Logical. If TRUE, the calibration maximizes the objective function.
cal_q_transfo	A string indicating how runoff should be transformed (see transfo_q).
lambda	Optional. A numeric value or vector used for regularization or transformation.
do_transfo_param	Logical. If TRUE, parameters are transformed from the unit hypercube to real-world values before simulation.
airGR_RunOptions	Optional. Run options for **airGR** models.
airGR_RunOptions_snow_module	Optional. Run options for the snow module if using **airGR**.

Details

The function handles parameter transformation, snow module simulation, and error handling for cases where the simulated runoff is invalid (e.g., all NAs or zeros).

Value

A single numeric value representing the error criterion to be minimized or maximized.

Note

- If Qsim is entirely NA, a large penalty value (+/- 1e10) is returned.
- If Qsim is all zeros and the criterion is KGE, the asymptotic value 1 - sqrt(3) is returned.
- Qsim is coerced to numeric to ensure compatibility with Qobs, especially for models like TUW.
- Future improvements may include spatially explicit versions and refined NA handling.

See Also

calibrate_model, create_input, load_meteo_data, calc_hydroGOF, transfo_param, simulate_model, simulate_snow

Examples

```
## Not run:
cal_results <- DEoptim::DEoptim(
  fn = optim_fn,
  lower = lower,
  upper = upper,
  control = DEoptim::DEoptim.control(NP = 50, itermax = 100),
  hydro_data = hydro_data,
  split_indices = split_indices,
  model = "GR4J",
  input = input,
  error_crit = "KGE",
  cal_maximize = TRUE,
  cal_q_transfo = "none",
  do_transfo_param = TRUE
)

## End(Not run)
```

---

save_airGR_plot *Save airGR Diagnostic Plots*

Description

Generates and saves diagnostic plots for hydrological model simulations using the **airGR** plotting functions. This function supports both airGR and non-airGR models by converting simulation results into a compatible format.

Usage

```
save_airGR_plot(
  file,
  model,
  simulation_results,
  ind = seq_along(simulation_results$date),
  hydro_data = NULL
)
```

Arguments

file	A string specifying the filename for the saved plot (e.g., "airGR_plot.pdf").
model	A string indicating the model name (e.g., "CemaNeigeGR4J"), used to determine if the model is from the airGR family. For a complete list see table in vignette("model_overview").
simulation_results	A list containing simulation outputs, typically from simulate_model. Must include at least Qsim, Qobs, and optionally snow-related variables like SWE, psolid, and pliquid.

ind	A vector of indices specifying the time period to plot. Defaults to the full time range.
hydro_data	Optional list containing BasinObs and BasinInfo, required if the model is not from the airGR family.

Details

If the model is not from the airGR family, the function reconstructs an airGR::OutputsModel object using create_input and simulate_model, and overrides it with the provided simulation results. If snow-related outputs are present, additional snow plots are included.

Value

A logical value indicating whether the plot was successfully saved.

Note

- Currently supports only one CemaNeige snow layer.
- A future version may support plotting without requiring Qobs.

See Also

simulate_model, plot, create_input

Examples

```
save_airGR_plot(
  file = "airGR_plot.pdf",
  model = "CemaNeigeGR4J",
  simulation_results = simulation_results,
  hydro_data = hydro_data
)
```

---

save_cal_val_plot *Save Calibration and Validation Plot*

Description

Creates and saves a plot comparing observed and simulated runoff over the calibration and validation periods. The plot includes performance metrics such as KGE, NSE, and percent bias, and displays them for each period.

Usage

```
save_cal_val_plot(file, BasinObs, Qsim, split_indices)
```

Arguments

`file`	A string specifying the filename for the saved plot (e.g., `"cal_val.pdf"`).
`BasinObs`	A data frame containing observed runoff and corresponding dates, typically from `load_meteo_data`. Must include columns `Qmm` and `DatesR`.
`Qsim`	A numeric vector of simulated runoff values.
`split_indices`	A list of indices from `split_data_set`, containing elements `ind_cal` and `ind_val` for calibration and validation periods, respectively.

Details

The function generates a two-panel plot showing observed and simulated runoff for both calibration and validation periods. It also computes and displays selected validation metrics using `calc_validation_results`. The metrics are shown as annotations on the plot.

Value

A logical value indicating whether the plot was successfully saved.

Note

Future improvements could include adding a seasonal rolling mean to the plot.

Examples

```
save_cal_val_plot(
  file = "cal_val.pdf",
  BasinObs = BasinObs,
  Qsim = simulation_results$Qsim,
  split_indices = split_indices
)
```

`set_airGR_par` *Create Default Calibration Settings for airGR Models*

Description

Generates a list of default calibration settings for a specified model from the airGR family. This includes parameter bounds and optimizer configurations for DEoptim, malschains, and hydroPSO.

Usage

```
set_airGR_par(model)
```

Arguments

`model`	A string specifying the airGR model (e.g., `"GR4J"`, `"CemaNeigeGR4J"`). For a complete list see table in `vignette("model_overview")`.

Details

The function uses `CreateCalibOptions` to extract parameter bounds and sets default configurations for several optimizers. It also detects whether the model includes a snow module based on the model name.

Value

A list containing:

- `lower`, `upper` – numeric vectors of parameter bounds.
- `nof_param` – number of parameters.
- `DEoptim`, `malschains`, `hydroPSO` – optimizer settings.
- `has_snow_module` – logical indicating whether the model includes a snow module.

Note

This function only supports models from the airGR family. An error is thrown otherwise.

Examples

```
set_airGR_par("GR4J")
set_airGR_par("CemaNeigeGR4J")
```

`set_cal_par` *Set Calibration Parameter Value*

Description

Updates a specific entry in a nested calibration parameter list for a given model.

Usage

```
set_cal_par(model, setting_name_value, cal_par)
```

Arguments

`model`	A string specifying the model name (e.g., `"TUW"`), used to access the corresponding sublist in `cal_par`. For a complete list see table in `vignette("model_overview")`.
`setting_name_value`	A string representing the setting to update, in the format `"sublist$parameter = value"`. This string is parsed and evaluated to modify the calibration settings.
`cal_par`	A list containing calibration settings for one or more models.

Details

This function uses non-standard evaluation to dynamically update a nested element in the `cal_par` list. It is useful for programmatically modifying calibration settings without manually navigating the list structure.

Value

The updated calibration settings list.

Examples

```
cal_par_updated <- set_cal_par("TUW", "DEoptim$itermax = 5", cal_par)
```

`set_hydromad_par` *Create Default Calibration Settings for hydromad Models*

Description

Generates a list of default calibration settings for a given hydromad model, including parameter bounds and optimizer configurations for DEoptim, malschains, and hydroPSO. Supports both SMA-only and SMA-routing model combinations.

Usage

```
set_hydromad_par(model, routing = "expuh")
```

Arguments

`model`	A string specifying the soil moisture accounting (SMA) model (e.g., `"gr4j"`, `"sacramento"`, `"snow"`), or a pre-defined `hydromad` model object. For a complete list see table in `vignette("model_overview")`.
`routing`	A string specifying the routing model (e.g., `"expuh"`, `"lambda"`). Only used if `model` is a string.

Details

If a model string is provided, a temporary hydromad model is created using the specified SMA and routing. Parameter bounds are extracted using `getFreeParsRanges`. Routing parameters are only added for models that support routing.

Value

A list containing:

- `lower`, `upper` – numeric vectors of parameter bounds.
- `nof_param` – number of parameters.
- `routing` – the routing model used.
- `DEoptim`, `malschains`, `hydroPSO` – optimizer settings.
- `has_snow_module` – logical indicating whether the model includes a snow module.

Examples

```
set_hydromad_par("gr4j")
set_hydromad_par("sacramento")
set_hydromad_par(hydromad(DATA = NULL, sma = "snow", routing = "expuh"))
```

`simulate_model` *Simulate a Hydrological Model*

Description

Simulates discharge using a specified hydrological model over a given time period. Supports models from the **airGR**, **TUWmodel**, **hydromad**, and **topmodel** packages. Optionally includes snow-related outputs if the model supports it.

Usage

```
simulate_model(
  model,
  model_param,
  input,
  ind = seq_along(input[[1]]),
  Qobs = NULL,
  airGR_RunOptions = NULL
)
```

Arguments

`model`	A string specifying the hydrological model to use. Supported models include `"TUW"`, `"topmodel"`, `"hydromad"` models (e.g., `"sacramento"`, `"cwi"`), and **airGR** models (e.g., `"GR4J"`, `"CemaNeigeGR4J"`). For a complete list see table in `vignette("model_overview")`.
`model_param`	A numeric vector of model parameters specific to the chosen model.
`input`	A list of model input data, typically created using `create_input`. Must include time series of precipitation (`P`), temperature (`T`), and optionally evapotranspiration (`E`), as well as spatial data like catchment area or topography.
`ind`	A vector of indices specifying the time steps to simulate. Defaults to the full range.
`Qobs`	Optional. A vector of observed discharge values to include in the output for comparison.
`airGR_RunOptions`	Optional. A pre-created `RunOptions` object for **airGR** models.

Value

A list containing:

- `date`: Vector of simulation dates.
- `Qsim`: Simulated discharge.
- `Qobs`: Observed discharge (if provided).
- `SWE`: Snow water equivalent (if available).
- `psolid`: Solid precipitation (if available).
- `pliquid`: Liquid precipitation (if available).
- `melt`: Meltwater (if available).
- `more_info`: A list with model-specific output.

Note

- For **airGR** models, the appropriate `RunModel_*` function is called.
- For **hydromad** models, missing dates are filled to ensure compatibility.
- Snow-related outputs are only available for models that simulate snow processes.
- If `Qobs` is provided, its length must match the length of `Qsim`.

See Also

`simulate_snow`, `create_input`, `calibrate_model`

Examples

```
## Not run:
simulation_results <- simulate_model(
  model = "TUW",
  model_param = calibration_results$model_param,
  input = input,
  ind = split_indices$ind_cal
)

## End(Not run)
```

---

simulate_snow *Simulate Snow Module*

Description

Simulates snow accumulation and melt processes using a specified snow module. Currently supports `"CemaNeige"` (from **airGR**) and `"TUWsnow"` (from **TUWmodel**). Returns surface water runoff and other snow-related variables such as SWE, melt, and precipitation components.

Usage

```
simulate_snow(
  snow_module,
  model_param,
  input,
  ind = seq_along(input[[1]]),
  airGR_RunOptions = NULL
)
```

Arguments

`snow_module`	A string specifying the snow module to use. Supported options are `"CemaNeige"` and `"TUWsnow"`. `"snowsnow"` is not yet implemented.
`model_param`	A numeric vector of snow module parameters.
`input`	A list containing input data for the snow module, including precipitation (`P`), air temperature (`T`), and optionally evapotranspiration (`E`).
`ind`	A vector of indices specifying the time steps to simulate. Defaults to the full time range.
`airGR_RunOptions`	Optional. A pre-created `RunOptions` object for **airGR** models.

Value

A list containing:

- `surface_water_runoff`: Simulated liquid water output (rain + melt).
- `SWE`: Snow water equivalent.
- `psolid`: Solid precipitation.
- `pliquid`: Liquid precipitation.
- `melt`: Meltwater.
- `more_info`: A list with additional model-specific output.

Note

- For `"CemaNeige"`, the function uses **airGR**'s `RunModel_CemaNeige`.
- For `"TUWsnow"`, the function uses **TUWmodel** and appends constant runoff parameters.
- The `"snowsnow"` module is not yet implemented due to missing runoff output.
- Multilayer or spatially distributed snowpack outputs are not yet supported.

See Also

`calibrate_model`, `simulate_model`, `RunModel_CemaNeige`, `TUWmodel`

---

split_data_set *Split a Date Vector into Warm-up, Calibration, and Validation Periods*

Description

Splits a date vector or a data frame with a date column into three time periods: warm-up, calibration, and validation. The function allows for optional adjustment of the calibration and validation periods based on the presence of missing data at the beginning of the time series.

Usage

```
split_data_set(
  df,
  start_end_date_vec,
  ensure_warm_up = TRUE,
  adjust_cal_end = FALSE,
  adjust_val_start = FALSE
)
```

Arguments

`df`	A vector of dates (e.g., `Date`, `POSIXt`) or a data frame containing a `DatesR` column and a `Qmm` column (used to detect the first non-NA value).
`start_end_date_vec`	A character vector of length six, specifying the start and end dates for the warm-up, calibration, and validation periods, in that order.
`ensure_warm_up`	Logical. If `TRUE`, adjusts the warm-up period to start at the first non-NA value in `Qmm`, if applicable. Default is `TRUE`.

`adjust_cal_end` Logical. If `TRUE`, the end date of the calibration period is adjusted proportionally to the shift in the warm-up period, preserving the original calibration-to-validation duration ratio. This ensures that the calibration period remains representative even if the warm-up period is shifted due to missing data.

`adjust_val_start` Logical. If `TRUE`, the start date of the validation period is adjusted to immediately follow the (potentially shifted) calibration period. This ensures continuity between calibration and validation periods when the calibration end date has been modified.

Value

A list with three elements:

ind_warm Indices corresponding to the warm-up period.

ind_cal Indices corresponding to the calibration period.

ind_val Indices corresponding to the validation period.

Examples

```
## Not run:
dates <- seq(as.Date("2000-01-01"), as.Date("2010-12-31"), by = "month")
df <- data.frame(DatesR = dates, Qmm = c(rep(NA, 12), runif(length(dates) - 12)))
periods <- split_data_set(df, c("2000-01-01", "2002-12-31", "2003-01-01", "2006-12-31", "2007-01-01", "2010-1

## End(Not run)
```

---

subset_simulations *Subset Simulation Results by Index*

Description

This function subsets the elements of a simulation results list to the specified indices. It excludes the `"more_info"` field by default but can optionally retain it.

Usage

```
subset_simulations(ind, simulation_results, keep_more_info = FALSE)
```

Arguments

`ind` A vector of indices to subset the simulation results.

`simulation_results` A list containing simulation results, where each element (except `"more_info"`) is assumed to be indexable.

`keep_more_info` Logical; if `TRUE`, the `"more_info"` field is retained in the output. Default is `FALSE`.

Details

This function assumes that all elements of `simulation_results`, except `"more_info"`, can be subset using the provided indices. A future enhancement could include a check to ensure that each field is indeed subsettable.

Value

A list of simulation results subset to the specified indices. If `keep_more_info` is `TRUE`, the `"more_info"` field is included unchanged.

Examples

```
## Not run:
results <- list(
  data = list(1:10, 11:20, 21:30),
  metrics = list(a = 1:3, b = 4:6),
  more_info = list(description = "Full simulation run")
)
subset_simulations(1:2, results, keep_more_info = TRUE)

## End(Not run)
```

---

transfo_param *Parameter Transformation to and from Hypercube*

Description

Transforms model parameters to the unit hypercube [0, 1] and back, depending on the specified direction. For `airGR` models, the corresponding transformation functions from the **airGR** package are used. For other models, a min-max normalization approach is applied using `norm_minmax`.

Usage

```
transfo_param(
  param,
  direction,
  model,
  snow_module = NULL,
  add_snow_par = FALSE,
  cal_parameter = default_cal_par
)
```

Arguments

`param`	A numeric vector of model parameters.
`direction`	A character string indicating the direction of transformation: `"RT"` (real to transformed) or `"TR"` (transformed to real).
`model`	A character string specifying the hydrological model (e.g., `"GR4J"`, `"TUW"`, `"CemaNeigeGR4J"`). For a complete list see table in `vignette("model_overview")`.

snow_module	Optional. A character string specifying the snow module to be included in the transformation (currently "CemaNeige" and "TUWsnow").
add_snow_par	Logical. If TRUE, snow module parameters are included in the transformation. Default is FALSE.
cal_parameter	A list containing calibration parameter bounds (lower and upper) for each model and snow module. Defaults to default_cal_par.

Details

Model combinations such as "CemaNeigeGR4J" are supported. Parameters for snow modules can also be included and transformed independently.

Value

A numeric vector of transformed or re-transformed parameters.

Note

1. CemaNeigeHyst is not yet implemented.
2. The airGR transformation functions require the **airGR** package to be installed.
3. Future versions may include an option to bypass **airGR** transformations entirely.

See Also

TransfoParam, norm_minmax

Examples

```
# Scale a parameter set for model "TUW" to [0,1] and back
param <- c(1, 2, 3, -1, 1, 1, 200, 10, 1, 15, 100, 50, 2, 15, 50)
scaled <- transfo_param(param, "RT", "TUW")
rescaled <- transfo_param(scaled, "TR", "TUW")
```

---

transfo_q *Apply transformation to runoff data*

Description

Supports inverse, square root, power, and Box-Cox transformations of runoff data. Log transformation is included but generally not recommended for KGE calculations (see airGR or Santos, 2018). Use Use KGEtang instead.

Usage

```
transfo_q(Q, q_transfo_type = "none", lambda = 0.25, ...)
```

Arguments

Q	A numeric vector, matrix, or data frame of runoff values.
q_transfo_type	A string specifying the transformation type. Options are: "none", "sqrt", "inv", "log", "boxcox", "boxcoxsantos", and "power".
lambda	A numeric value used for Box-Cox and power transformations. Default is 0.25.
...	Additional arguments passed to mean(), e.g., na.rm = TRUE, used in boxcoxsantos.

Value

Transformed runoff data in the same format as input.

Note

Consider how to handle infinite values resulting from transformations.

Examples

```
transfo_q(array(0:10, c(2, 5)), "sqrt")
```

---

validate_model *Validate model*

Description

Calculates validation measures for different transformation types

Usage

```
validate_model(Qsim, Qobs, val_crit_transfo = "KGE__none")
```

Arguments

Qsim	vector with simulated runoff
Qobs	vector with observed runoff
val_crit_transfo	a vector of strings specifying validation criteria and a runoff transformation separated by a "__". Supported are validation criteria from the hydroGOF package usable by the calc_hydroGOF function, for supported runoff transformations please refer to transfo_q

Value

a long data frame with columns crit indicating the used validation criterion, transfo for the used runoff transformation and value.

Examples

```
validate_model(
  1:10, seq(2, 11),
  c("KGE__log", "NSE__inv", "VE__none", "pbias__none")
)
```

`write_ascii` *Write ASCII Summary of Calibration and Validation Results*

Description

Writes a plain text (ASCII) file summarizing model calibration parameters and validation results. The output can be written in either tab-separated or fixed-width format.

Usage

```
write_ascii(
  file,
  calibration_results,
  validation_results,
  equally_spaced = TRUE
)
```

Arguments

`file`	A string specifying the path to the output file.
`calibration_results`	A list containing the calibration results from `calibrate_model`. Only the vector of calibrated model parameters (`model_param`) is written.
`validation_results`	A data frame containing validation results, typically from `validate_model`.
`equally_spaced`	Logical; if `TRUE` (default), attempts to write a fixed-width formatted file. If `FALSE` or if fixed-width writing fails, a tab-separated file is written instead.

Details

If `equally_spaced = TRUE`, the function attempts to write a fixed-width formatted file using `write.fwf`. If an error occurs during this process, it falls back to writing a tab-separated file using `write_tsv`.

Value

A logical value indicating whether the file was successfully written.

Examples

```
write_ascii(
  file = "results.txt",
  calibration_results = calibration_results,
  validation_results = validation_results
)
```

Index