Package: OptirrigTOOLS (via r-universe)

May 23, 2026

Type Package

Title Tools to Analyse and Visualise Data from the OptirrigHIVE Platform

Version 0.1.0

Description Provides tools to prepare, analyse, and visualise data from the OptirrigHIVE platform. The package includes utilities for scenario building, data transformation, statistical analysis, and irrigation- oriented visualisation workflows for OptirrigCORE studies.

Encoding UTF-8

License AGPL (>= 3)

BugReports https://forge.inrae.fr/OptirrigHIVE/OptirrigTOOLS/-/issues

URL https://forge.inrae.fr/OptirrigHIVE/OptirrigTOOLS

LazyData true

Depends R (>= 4.1.0)

Imports utils, stats, cli, dplyr, ggplot2, ini, logger, magrittr, OptirrigCORE, patchwork, readr, rlang, scales, stringr, tibble, tidyr

Suggests DEoptim, magick, neldermead, styler, testthat (>= 3.0.0), knitr, rmarkdown

Additional_repositories https://inrae.r-universe.dev

RoxygenNote 7.3.3

Config/testthat/edition 3

Roxygen list(markdown = TRUE)

Config/pak/sysreqs libicu-dev libx11-dev

Repository https://inrae.r-universe.dev

Date/Publication 2026-04-23 12:45:47 UTC

RemoteUrl https://forge.inrae.fr/OptirrigHIVE/OptirrigTOOLS

RemoteRef main

RemoteSha 754d119e06836ad34614926ce7d5ae4490d554e4

Contents

add_input_files_to_add

Add Input Files to Project Inputs Directory

Description

Copies specified input files into the "inputs" directory of a given project.

Usage

```
add_input_files_to_add(x, project_name, input_files_to_add, ...)
```

Arguments

`x`	character. Path to the base directory where the project is located.
`project_name`	character. Name of the project folder.
`input_files_to_add`	character. Paths to the input files to be copied. If `NULL`, no files are copied.
`...`	Additional arguments.

Details

This function copies the files specified in `input_files_to_add` into the "inputs" subdirectory of the project directory (`file.path(x, project_name, "inputs")`). If `input_files_to_add` is `NULL`, the function does nothing.

Value

Invisibly returns `x`.

build_uncertainty_mc_global_panels

Build MC global uncertainty panels (no OAT)

Description

Build MC global uncertainty panels (no OAT)

Usage

```
build_uncertainty_mc_global_panels(
  x,
  show_quantile_lines = TRUE,
  show_median_line = TRUE,
  x_axis = c("Date", "TT_sowing"),
  tt_step = 20
)
```

Arguments

x	Output of prepare_uncertainty_mc_plot_data(), result from run_uncertainty_mc(), or path to an .rds file.
show_quantile_lines	Logical; if TRUE, draw q05/q25/q75/q95 lines.
show_median_line	Logical; if TRUE, draw median line.
x_axis	Axis used for daily uncertainty ribbons: "Date" or "TT_sowing".
tt_step	Thermal-time grid step used when x_axis = "TT_sowing".

Value

Named list of ggplot objects (climate, water, plant, seasonal).

---

calc_criteria *Calculate Performance Metrics for Model Evaluation*

Description

This function computes various performance metrics to evaluate the predictive accuracy of a model. By comparing observed (obs) and simulated (sim) data, it provides insights into how well the simulated data matches the observed values. The function supports widely used metrics like Nash-Sutcliffe Efficiency (NSE), Root Mean Square Error (RMSE), Mean Absolute Error (MAE), and others. Normalized versions of error metrics (nRMSE, nMAE) are also available, enabling comparisons across datasets with different scales.

Usage

```
calc_criteria(obs, sim, error = NULL, weights = NULL, method, norm = "mean")
```

Arguments

obs	A numeric vector representing the observed (actual) data. This should contain the true values that the model attempts to predict.
sim	A numeric vector representing the simulated (predicted) data. These are the values produced by the model being evaluated.
error	(Optional) A numeric vector representing uncertainty in the observed values. If provided, the function adjusts calculations to account for error bounds.
weights	(Optional) Numeric vector of point weights (same length as obs/sim). If omitted, equal weights are used.
method	A string specifying the performance metric to calculate. Supported options include: • "NSE": Nash-Sutcliffe Efficiency • "RMSE": Root Mean Square Error

- `"pRMSE"` / `"prmse"`: Pondered RMSE alias
- `"MAE"`: Mean Absolute Error
- `"nRMSE"`: Normalized Root Mean Square Error
- `"nMAE"`: Normalized Mean Absolute Error
- `"R2"`: Coefficient of Determination (R-squared)
- `"dWillmot"`: Index of Agreement

`norm` A string indicating the normalization method for normalized metrics (`"nRMSE"` and `"nMAE"`). Supported options:

- `"mean"`: Normalize by the mean of `obs` (default).
- `"sd"`: Normalize by the standard deviation of `obs`.
- `"mxmn"`: Normalize by the range (max - min) of `obs`.
- `"iq"`: Normalize by the interquartile range (IQR) of `obs`.

Details

Supported Metrics::

- **Nash-Sutcliffe Efficiency (NSE)**: Measures how well the simulated data matches observed data. A value of 1 indicates perfect agreement, while values <0 suggest poor performance.

$$NSE = 1 - \frac{\sum (obs_i - sim_i)^2}{\sum (obs_i - \overline{obs})^2}$$

- **Root Mean Square Error (RMSE)**: Indicates the standard deviation of prediction errors. Smaller values suggest better performance.

$$RMSE = \sqrt{\frac{\sum (obs_i - sim_i)^2}{n}}$$

- **Mean Absolute Error (MAE)**: Calculates the mean of absolute differences between observed and simulated values.

$$MAE = \frac{\sum |obs_i - sim_i|}{n}$$

- **Normalized RMSE (nRMSE)**: A normalized version of RMSE based on the chosen normalization method.

$$nRMSE = \frac{RMSE}{\text{norm}(obs)}$$

- **Normalized MAE (nMAE)**: Similar to nRMSE but applies normalization to MAE.

$$nMAE = \frac{MAE}{\text{norm}(obs)}$$

- **Coefficient of Determination (R2)**: Measures the proportion of variance in `obs` explained by `sim`.

$$R^2 = \left(\frac{\text{cov}(obs, sim)}{\sigma_{obs} \sigma_{sim}} \right)^2$$

- **Index of Agreement (d)**: Indicates the level of predictive accuracy, with values ranging from 0 (no agreement) to 1 (perfect agreement).

$$d = 1 - \frac{\sum (obs_i - sim_i)^2}{\sum (|sim_i - \overline{sim}| + |obs_i - \overline{obs}|)^2}$$

Value

A numeric value representing the calculated performance metric based on the specified method. Higher efficiency metrics (e.g., NSE, R2, d) and lower error metrics (e.g., RMSE, MAE) indicate better model performance.

Note

If `obs` or `sim` contain only NA values, the function returns `NA`. Similarly, if `obs` has only a single non-NA value, relative error metrics are computed.

References

- Willmott, C.J., "On the Validation of Models", Physical Geography, 1981. DOI:10.1080/02723646.1981.10642213
- Willmott, C.J., "Some Comments on the Evaluation of Model Performance", Bulletin of the American Meteorological Society, 1982. DOI:10.1175/1520-0477(1982)063<1309:SCOTEO>2.0.CO;2

Examples

```
# Example of observed and simulated data
obs <- c(1, 2, 3, 4, 5)
sim <- c(1.1, 1.9, 3.1, 4.2, 4.8)

# Calculate Index of Agreement (d)
calc_criteria(obs, sim, method = "dWillmot")

# Calculate Nash-Sutcliffe Efficiency (NSE)
calc_criteria(obs, sim, method = "NSE")

# Calculate normalized RMSE (nRMSE)
calc_criteria(obs, sim, method = "nRMSE", norm = "mean")
```

---

calib_build_inputs_all

Build model inputs for multiple calibration trials

Description

Creates one Optirrig scenario per trial by taking fixed inputs directly from `datasets` and injecting values from `best_par` on top.

Usage

```
calib_build_inputs_all(datasets, trials = NULL, best_par)
```

Arguments

datasets	Named list of trial datasets.
trials	Optional character vector of trial names. If NULL, all dataset names are used.
best_par	Named parameter vector.

Value

A flat list of model inputs suitable for run_model().

---

calib_build_specs *Build multi-trial calibration specs*

Description

Wrapper around create_spec() to create one spec per trial.

Usage

```
calib_build_specs(
  datasets,
  trials = NULL,
  params = NULL,
  ranges,
  project_dir,
  yield_var = "Y_sugar",
  date_start = "01/01",
  date_end = "31/12"
)
```

Arguments

datasets	Named list of trial datasets.
trials	Character vector of trial names. If NULL, uses calib_get_trials_from_datasets().
params	params used in base overrides.
ranges	Parameter ranges for calibration.
project_dir	Project directory used by calibration pipeline.
yield_var	Yield variable passed to create_spec().
date_start	Calibration window start (MM/DD).
date_end	Calibration window end (MM/DD).

Value

Named list of specs.

calib_display_helpers *Calibration display helpers*

Description

Utility functions used to parse calibration outputs, reshape observed and simulated values, compute relative-deviation summaries, and assemble the diagnostic plots used in calibration workflows.

Usage

```
calib_display_parse_num(x)

calib_display_parse_date(x)

calib_display_get_run_id(i, inputs, outputs)

calib_display_get_trial_type(run_id)

calib_display_get_site_label(run_id)

calib_display_build_sim_long(outputs, inputs, vars, x_var = "TT_sowing")

calib_display_build_obs_long(inputs, outputs, vars, x_var = "TT_sowing")

calib_display_make_palette()

calib_display_plot_obs_vs_sim(obs_sim, title = NULL)

calib_display_plot_obs_sim_curves(
  sim_df,
  obs_df,
  title = NULL,
  x_var = "TT_sowing"
)

calib_display_plot_trials_grid(
  sim_df,
  obs_df,
  var,
  var_pot,
  trial_levels,
  title = NULL,
  which_site_ids = NULL,
  x_var = "TT_sowing",
  uncertainty_df = NULL
)
```

```
calib_display_calc_ed(a, b, method = "classic")

calib_display_summarize_ed(ed, stat = "mean")

calib_display_compute_ed_summary(
  obs_sim,
  var = "Y_sugar",
  method = "classic",
  stat = "mean",
  trial_order_ed = c("Ndc", "Irri", "ArrT", "ArrP", "IrriMn", "DclT", "Tem")
)

calib_display_plot_ed_recap(df_ed, title = NULL, which_id = NULL, ncol = 3L)

calib_display_get_top_screen(screen_res, param_cols, n = 10L)

calib_display_run_once(
  datasets,
  best_par,
  display_dir,
  pdf_name,
  title_prefix = "Optimized params",
  trials_display = NULL,
  vars_main = c("B", "LAI", "Y_sugar"),
  trial_levels = c("Ndc", "Irri", "ArrT", "ArrP", "IrriMn", "DclT", "Tem"),
  trial_order_ed = trial_levels,
  x_var = "TT_sowing",
 run_options = list(log = FALSE, save = FALSE, force = TRUE, return_outputs = TRUE, plan
    = "multicore", workers = 10),
  uncertainty_daily = NULL,
  png_width = 12,
  png_height = 9.86,
  png_dpi = 160
)
```

Arguments

x	Object to parse as numeric values.
i	Index of the run to inspect.
inputs	List of model input objects.
outputs	List of model output tables.
run_id	Run identifier.
vars	Character vector of variable names to extract.
x_var	X-axis variable to use, usually "TT_sowing" or "Date".
obs_sim	Data frame joining observed and simulated values.
title	Optional plot title.

sim_df	Long simulated data frame built by the calibration display helpers.
obs_df	Long observed data frame built by the calibration display helpers.
var	Variable to analyse or plot.
var_pot	Potential counterpart to var used in trial-grid plots.
trial_levels	Character vector giving the display order of trial types.
which_site_ids	Optional subset of site identifiers to display.
uncertainty_df	Optional uncertainty summary used to add q05 / q95 ribbons in grid plots.
a	Numeric vector for the first treatment or series in relative deviation calculations.
b	Numeric vector for the second treatment or series in relative deviation calculations.
method	Relative-deviation method.
ed	Numeric vector of relative deviations.
stat	Summary statistic to apply to relative deviations.
trial_order_ed	Trial order used to normalize pair ordering for relative-deviation summaries.
df_ed	Relative-deviation summary table produced by the calibration display helpers.
which_id	Optional subset of IDs to display in recap plots.
ncol	Number of columns in recap facets.
screen_res	Screening result object containing optim$results.
param_cols	Character vector of parameter column names.
n	Number of rows to keep from screening results.
datasets	Calibration datasets used to rebuild model inputs.
best_par	Named vector or list of best parameter values.
display_dir	Output directory used to save plots.
pdf_name	Base file name associated with exported plots.
title_prefix	Title prefix shared across generated plots.
trials_display	Optional subset of trial identifiers to display.
vars_main	Main observed variables to display.
run_options	Options passed to OptirrigCORE::run_model().
uncertainty_daily	Optional daily uncertainty summary used as ribbons in trial-grid plots.
png_width	Plot export width in inches.
png_height	Plot export height in inches.
png_dpi	Plot export resolution in dpi.

Value

Depending on the helper, a parsed vector, a long-format data frame, a ggplot object, a numeric summary, or a list bundling generated diagnostics.

calib_get_trials_from_datasets

Get valid calibration trials from datasets

Description

Keeps only trials that contain at least `min_obs` observed points for one variable in `trial$obs`.

Usage

```
calib_get_trials_from_datasets(datasets, min_obs = 3L)
```

Arguments

`datasets`	Named list of trial datasets.
`min_obs`	Minimum number of observed points required per variable.

Value

Character vector of trial names to calibrate.

calib_plot_main_trials

Plot main calibration diagnostics for all trials

Description

Plot main calibration diagnostics for all trials

Usage

```
calib_plot_main_trials(
  outputs_all,
  inputs_all,
  eval_all,
  title_prefix = "All trials",
  x_var = c("TT_sowing", "Date")
)
```

Arguments

`outputs_all`	Model outputs list.
`inputs_all`	Model inputs list.
`eval_all`	Evaluation list containing `obs_sim`.
`title_prefix`	Prefix used in plot titles.
`x_var`	X axis for curves/grids (`"TT_sowing"` or `"Date"`).

Value

Invisibly returns NULL.

`calib_print_eval_summary` *Print calibration score summary*

Description

Print calibration score summary

Usage

```
calib_print_eval_summary(eval_all, scores_all_trials)
```

Arguments

`eval_all`	Output from `run_model_eval()`.
`scores_all_trials`	Trial-level score table from `calib_run_and_eval_trials()`.

Value

Invisibly returns `scores_all_trials`.

`calib_run_and_eval_trials` *Run model and compute trial-level calibration scores*

Description

Runs all inputs, evaluates Obs/Sim with `run_model_eval()`, then computes one score per run and variable, plus a mean score per run.

Usage

```
calib_run_and_eval_trials(
  inputs_all,
  trials,
 run_options = list(log = FALSE, plan = "multicore", workers = 1, force = TRUE, save =
    FALSE, return_outputs = TRUE),
  method = "nrmse"
)
```

Arguments

inputs_all	List of model inputs.
trials	Character vector of trial names.
run_options	Options list passed to OptirrigCORE::run_model().
method	Criterion used for evaluation and score computation.

Value

List with outputs_all, eval_all, scores_all_trials.

---

create_project *Create a new project directory structure with optional crop, soil, and input files.*

Description

This function initializes a new project by creating the main directory and adding optional crop and soil information. It also generates parameter CSV and user configuration files, and can add additional input files if specified.

Usage

```
create_project(
  project_name,
  crop = NULL,
  soil = NULL,
  path = system.file("extdata", package = pkg),
  pkg = utils::packageName(),
  overwrite = FALSE,
  input_files_to_add = NULL,
  log = TRUE,
  ...
)
```

Arguments

project_name	character. Name of the project to create.
crop	character. Crop information to include in the project.
soil	character. Soil information to include in the project.
path	character. Path where the project will be created. Defaults to the "extdata" directory of the package.
pkg	character. Name of the package. Defaults to the current package name.
overwrite	logical. Whether to overwrite an existing project directory. Defaults to FALSE.

input_files_to_add	character. Additional input files to add to the project.
log	logical. Whether to log the creation of the project. Defaults to TRUE.
...	Additional arguments passed to lower-level directory creation functions.

Details

The function creates a project directory structure, adds parameter and configuration files, and optionally copies additional input files. Logging is performed if enabled.

Value

Invisibly returns the path to the created project directory.

---

create_project.config_user

Create a User Configuration File for a Project

Description

This function generates a user-specific configuration file (`config.user.ini`) for a given project. It creates a directory structure for the project if it does not exist and writes a configuration file containing project metadata, data paths, and directory paths.

Usage

```
create_project.config_user(
  x,
  project_name,
  pkg = utils::packageName(),
  version = NULL,
  ...
)
```

Arguments

x	character The base directory path where the project will be created.
project_name	character The name of the project.
pkg	character The package name to retrieve the version from. Defaults to the current package.
version	character The version of the project or package. If `NULL`, attempts to retrieve the version from the package.
...	Additional arguments.

Details

The function creates a configuration list with project name and version, data path, and standard subdirectories for configuration, inputs, and outputs. It writes this configuration to an INI file using the `ini` package.

Value

Invisibly returns the input `x`.

`create_project.dir` *Create a Project Directory Structure and Copy Configuration Files*

Description

This function creates a standardized directory structure for a new project, including subdirectories for configuration, input, and output files. It also copies relevant `.ini` configuration files from a specified package's resource directory into the project's configuration folder. Optionally, crop- and soil-specific `.ini` files can be included.

Usage

```
create_project.dir(x, project_name, crop, soil, pkg, overwrite = TRUE, ...)
```

Arguments

`x`	character. Path to the parent directory where the project will be created.
`project_name`	character. Name of the new project directory.
`crop`	character. Crop identifier to include crop-specific `.ini` files. Default is NULL.
`soil`	character. Soil identifier to include soil-specific `.ini` files. Default is NULL.
`pkg`	character. Name of the package containing the resource `.ini` files.
`overwrite`	logical. Whether to overwrite existing files. Default is TRUE.
`...`	Additional arguments.

Details

The function creates the following subdirectories within the project directory:

- `config`
- `inputs`
- `outputs`

It copies all `.ini` files from the package resource directory, excluding those with `.user.`, `.plant.`, or `.soil.` in their names, unless they match the specified crop or soil.

Value

Invisibly returns the path to the parent directory (`x`).

`create_project.params_csv` *Create a Project Parameters CSV File*

Description

This function generates a CSV file containing project parameters for a given project. It reads the default parameters from a `.run.ini` file within the specified package, processes them, and writes them to a CSV file in the project directory. If the CSV file already exists, the function does nothing.

Usage

```
create_project.params_csv(x, project_name, pkg = utils::packageName(), ...)
```

Arguments

`x`	character. Path to the base directory where the project folder is located.
`project_name`	character. Name of the project; used to create the project folder and CSV file.
`pkg`	character. Name of the package containing the `.run.ini` file. Defaults to the current package.
`...`	Additional arguments.

Details

The function searches for a `.run.ini` file in the specified package, reads its contents, and processes the parameters into a data frame. It removes columns matching the pattern `"^types\\."`, adds columns for `run_id`, `crop`, `soil`, `date_sowing`, and `date_harvest`, and relocates some columns for convenience. The resulting data frame is written to a CSV file named `"<project_name>_params.csv"` inside the project directory.

Value

Invisibly returns the input path `x`.

`create_scenarios` *Create model scenarios from category blocks (grid expansion or automatic zip)*

Description

`create_scenarios()` assembles model parameters provided by category blocks (`inputs`, `crop`, `soil`, `irrigation`, `run`) into one or more structured scenarios (class `"model_scenarios"`).

Usage

```
create_scenarios(
  inputs = list(),
  crop = list(),
  soil = list(),
  irrigation = list(),
  run = list(),
  options = list(grid = FALSE, fill_missing = FALSE)
)
```

Arguments

`inputs`	Named list of observed inputs (e.g. climate, irrigation calendar).
`crop`	Named list of crop parameters.
`soil`	Named list of soil parameters.
`irrigation`	Named list of irrigation parameters (multi-period values as `list()`).
`run`	Named list of run parameters (`date_start`, `date_end`, `run_id`, ...).
`options`	List of options: **grid** Logical. If `TRUE`, generate a cartesian product of parameter values. **fill_missing** Logical. If `TRUE`, call `OptirrigCORE::create_model_inputs()` to materialize defaults/loaders.

Details

The function supports two scenario generation modes:

Grid mode (`options$grid = TRUE`):
Produces the cartesian product of varying parameters (all combinations), using `expand_grid()` on the flattened configuration. This is appropriate for factorial sensitivity analyses (e.g., exploring all combinations of `zmax` and `ratio_R_init`).

Automatic zip mode (`options$grid = FALSE`, default):
Produces *N* scenarios by aligned indexing: first values together, second values together, etc. This mode is designed to safely create multi-run batches without exploding to a full grid.

Zip triggers:
When `grid = FALSE`, scenario count (*N*) is driven **only** by atomic vectors (numeric/character/logical) with `length > 1` found in the safe blocks: `run`, `crop`, and `soil`. The `inputs` and `irrigation` blocks are *not* zip triggers, to avoid accidentally creating scenarios from naturally vector inputs (time series, calendars, constraints vectors, etc.).

Irrigation slicing in multi-run zip:
Even though irrigation values do not trigger zipping, when multiple scenarios are created (`N > 1`), irrigation parameters can be supplied per-run and will be sliced by index if:

- they are atomic vectors of length `N`, or
- they are lists (non-data.frame) of length `N`.

`irrig_not_allowed` is explicitly excluded from run-wise slicing (it is assumed to be an intra-run vector of dates, not a per-run vector).

Length sanity checks (zip mode):
If `grid = FALSE` and `N > 1`, all `inputs` parameters that are atomic vectors or lists must have length `1` or `N`. For `irrigation`, lists of scalar values (e.g. constraint windows) are treated as single values and do not need to match `N`. Data frames and matrices are ignored by this check (they are treated as single objects).

Run ID normalization:

If multiple scenarios are produced and `run_id` is missing, a fallback ID is assigned (`"Run_1"`, `"Run_2"`, ...). After scenario construction, if any run IDs are missing or duplicated, IDs are replaced deterministically with `"Run_1" .. "Run_n"`, and scenario list names are aligned accordingly.

Filling missing inputs:

If `options$fill_missing = TRUE`, the returned scenarios are fully materialized using `OptirrigCORE::create_model_inp` and `OptirrigCORE::load_config()` (defaults + parsing + loaders), making them directly runnable with `OptirrigCORE::run_model()`.

Value

An object of class `"model_scenarios"` (a named list of scenarios).

Examples

```
## Not run:
devtools::load_all()
library(OptirrigCORE)
library(patchwork)

# ------------------------------------------------------------------------
# Test run_model with an input CSV file (single project)
# ------------------------------------------------------------------------
path <- system.file("extdata", package = "OptirrigCORE")
project_name <- "2021_mais_Lav"
input <- read(file.path(path, project_name, "2021_mais_Lav_params_no_irrig.csv"))
input_path <- file.path(path, project_name, "inputs")

outputs <- run_model(
  input,
  options = list(log = FALSE),
  cfg = load_config(path_directory = input_path)
)
plot_outputs(outputs)

# ------------------------------------------------------------------------
# Load example climate + irrigation objects used in the dev workflow
# ------------------------------------------------------------------------
data("2021clim")
data("2021irrig")

# ------------------------------------------------------------------------
# Case 1 - 1 simulation (zip mode, but no multi-length parameters in run/crop/soil)
```

```
# ------------------------------------------------------------------------
scen_1 <- create_scenarios(
  run = list(
    run_id = "Run_1",
    date_start = "01/04/2021",
    date_end = "06/09/2021"
  ),
  inputs = list(
    climate = climate,
    irrigation = NA
  ),
  crop = list(
    crop = "maize",
    date_sowing = "20/04/2021",
    date_harvest = "06/09/2021"
  ),
  soil = list(
    soil = "CL",
    zmax = 1.2,
    ratio_R_init = 70
  ),
  irrigation = list(
    irrig_mod = 1,
    tbis_mod = 1,
    TT_d = 750,
    dose_fix = 10,
    irrig_strategy_start = "25/04/2021",
    irrig_strategy_end = "01/08/2021",
    irrig_constraints_start = "01/06/2021",
    irrig_constraints_end = "01/08/2021",
    irrig_not_allowed = c("11/06/2021", "17/06/2021", "03/07/2021", "19/07/2021"),
    WTurn_mod = 1,
    WTurn_ndays = 1,
    RU_thrld = list(30, 70),
    RU_target = list(30, 90),
    RU_units = 2,
    dose_max = 20,
    quota_max = 400,
    quota_wind = 20
  ),
  options = list(grid = FALSE, fill_missing = TRUE)
)

outputs_1 <- run_model(
  x = scen_1,
  options = list(plan = "sequential", log = FALSE)
)
plot_outputs(outputs_1)

# ------------------------------------------------------------------------
# Case 2 - 5 simulations without grid (automatic zip)
# zmax and ratio_R_init are vectors in the soil block => N = 5 runs (aligned)
# ------------------------------------------------------------------------
```

```
scen_5 <- create_scenarios(
  run = list(
    run_id = "Run_1",
    date_start = "01/04/2021",
    date_end = "06/09/2021"
  ),
  inputs = list(climate = climate, irrigation = NA),
  crop = list(crop = "maize", date_sowing = "20/04/2021", date_harvest = "06/09/2021"),
  soil = list(
    soil = "CL",
    zmax = seq(1.2, 2, 0.2),
    ratio_R_init = seq(50, 90, 10)
  ),
  irrigation = list(
    irrig_mod = 1,
    tbis_mod = 1,
    TT_d = 750,
    irrig_strategy_start = "25/04/2021",
    irrig_strategy_end = "01/08/2021",
    irrig_constraints_start = list("25/04/2021", "25/06/2021"),
    irrig_constraints_end = list("25/06/2021", "01/08/2021"),
    irrig_not_allowed = c("11/06/2021", "17/06/2021", "03/07/2021", "19/07/2021"),
    WTurn_mod = 1,
    WTurn_ndays = 5,
    RU_thrld = list(30, 70),
    RU_target = list(30, 90),
    RU_units = 2,
    dose_fix = list(15, 30),
    quota_max = 400
  ),
  options = list(grid = FALSE, fill_missing = TRUE)
)

outputs_5 <- run_model(
  x = scen_5,
  options = list(plan = "multicore", log = FALSE),
  cfg = load_config()
)
plot_outputs(outputs_5)

# ------------------------------------------------------------------------
# Case 3 - 25 simulations with grid (full factorial)
# zmax (5 values) x ratio_R_init (5 values) => 25 combinations
# ------------------------------------------------------------------------
scen_25 <- create_scenarios(
  run = list(run_id = "Run_1", date_start = "01/04/2021", date_end = "06/09/2021"),
  inputs = list(climate = climate, irrigation = NA),
  crop = list(crop = "maize", date_sowing = "20/04/2021", date_harvest = "06/09/2021"),
  soil = list(soil = "CL", zmax = seq(1.2, 2, 0.2), ratio_R_init = seq(50, 90, 10)),
  irrigation = list(
    irrig_mod = 1, tbis_mod = 1, TT_d = 750,
    irrig_strategy_start = "25/04/2021",
    irrig_strategy_end = "01/08/2021",
```

```
    irrig_constraints_start = list("25/04/2021", "25/06/2021"),
    irrig_constraints_end = list("25/06/2021", "01/08/2021"),
    irrig_not_allowed = c("11/06/2021", "17/06/2021", "03/07/2021", "19/07/2021"),
    WTurn_mod = 1, WTurn_ndays = 5,
    RU_thrld = list(30, 70), RU_target = list(30, 90), RU_units = 2,
    dose_fix = list(15, 30),
    quota_max = 400
  ),
  options = list(grid = TRUE, fill_missing = TRUE)
)

outputs_25 <- run_model(
  x = scen_25,
  options = list(plan = "multicore", log = FALSE),
  cfg = load_config()
)
plot_outputs(outputs_25)

# ------------------------------------------------------------------------
# Case 4 - 10 simulations with explicit irrigation strategies
# run_id is a vector => zip creates 10 runs, and irrigation fields are sliced
# when length == 10 (atomic) or list length == 10
# ------------------------------------------------------------------------
scen_strat <- create_scenarios(
  run = list(
    run_id     = c("S1", "S2", "S3", "S4", "S5", "S6", "S7", "S8", "S9", "S10"),
    date_start = "01/04/2021",
    date_end   = "06/09/2021"
  ),
  inputs = list(
    climate = climate,
    irrigation = list(NA, irrig, NA, NA, NA, NA, NA, NA, NA, NA)
  ),
  crop = list(
    crop = "maize",
    date_sowing = "20/04/2021",
    date_harvest = "06/09/2021"
  ),
  soil = list(
    soil = "CL",
    zmax = 1.2,
    ratio_R_init = 80,
    theta_fc = 0.30,
    theta_wp = 0.17,
    theta_sat = 0.42,
    theta_raw = 0.21,
    theta_r = 0.04
  ),
  irrigation = list(
    irrig_mod = c(0, 0, 1, 1, 1, 1, 1, 1, 1, 1),
    irrig_strategy_start = c(
      rep(list("05/05/2021"), 8),
      list(c("05/05/2021", "01/07/2021")),
```

```
      list(c("05/05/2021", "01/07/2021"))
    ),
    irrig_strategy_end = c(
      rep(list("08/08/2021"), 8),
      list(c("01/07/2021", "08/08/2021")),
      list(c("01/07/2021", "08/08/2021"))
    ),
   irrig_constraints_start = c(NA, NA, NA, NA, NA, NA, NA, NA, "01/07/2021", "10/07/2021"),
    irrig_constraints_end = c(NA, NA, NA, NA, NA, NA, NA, NA, "20/07/2021", "20/07/2021"),
    RU_thrld = list(NA, NA, 25, 50, 35, NA, 75, 75, list(30, 70), list(35, 50)),
    RU_target = list(NA, NA, 75, 80, 60, NA, 100, 100, list(30, 70), list(75, 65)),
    PI_thrld = c(NA, NA, NA, NA, NA, 30, NA, NA, NA, NA),
    PI_ndays = c(NA, NA, NA, NA, NA, 15, NA, NA, NA, NA),
    WTurn_ndays = c(NA, NA, NA, 7, NA, 7, NA, 7, 5, NA),
    WTurn_mod = c(NA, NA, 0, 1, 0, 1, 0, 1, 2, 0),
    dose_min = c(NA, NA, NA, NA, NA, NA, NA, 15, NA, 15),
    dose_max = c(NA, NA, NA, NA, NA, NA, NA, 20, NA, 20),
    dose_fix = c(NA, NA, 30, 30, 30, 20, 30, 30, 20, 30),
    quota_max = c(NA, NA, NA, NA, 400, NA, 300, 200, NA, 300),
    quota_wind = c(NA, NA, NA, NA, NA, NA, NA, NA, 100, 60)
  ),
  options = list(grid = FALSE, fill_missing = FALSE)
)

outputs_strat <- run_model(
  x = scen_strat,
  options = list(plan = "multicore", log = FALSE),
  cfg = load_config()
)
plot_outputs(
  outputs = outputs_strat,
  options = list(
    run_prefix = "S",
    pos = list("RU | Y", "IRR | B"),
    title = "Optirrig - Irrigations Strategies"
  )
)

## End(Not run)
```

`create_spec` *Build a project specification for uncertainty/calibration workflows*

Description

Build a spec object consumable by calibration and uncertainty helpers.

Usage

```
create_spec(
```

```
  project_name,
  ranges,
  climate = NULL,
  irrigation = NULL,
  obs = NULL,
  crop = NULL,
  soil = NULL,
  date_sowing = NULL,
  date_harvest = NULL,
  date_start = date_sowing,
  date_end = date_harvest,
  irrig_mod = 0,
  yield_var = "Y",
  base_overrides = list(),
  optirrig_data = NULL,
  optirrig_crop_map = c(sugarbeet = "beetroot")
)
```

Arguments

`project_name`	Character scalar. Project folder name.
`ranges`	Named list of parameter ranges (`c(lower, upper)` or `c(lower, upper, decimals)`).
`climate`	Climate table used by the model.
`irrigation`	Irrigation table used by the model.
`obs`	Optional named list of observed series specs.
`crop`	Character scalar. Crop name used by the model.
`soil`	Character scalar. Soil identifier used by the model.
`date_sowing`	Character/Date sowing date.
`date_harvest`	Character/Date harvest date.
`date_start`	Character/Date simulation start date.
`date_end`	Character/Date simulation end date.
`irrig_mod`	Numeric scalar irrigation mode modifier.
`yield_var`	Character scalar (`"Y"` or `"Y_sugar"`).
`base_overrides`	Named list to override/add base fields.
`optirrig_data`	Optional simplified payload from `OptirrigDatasets::load_datasets(..., simplify = TRUE)[[trial]]`.
`optirrig_crop_map`	Named mapping from dataset crop ids to model crop ids.

Value

A list with `project_name`, normalized `ranges`, and `base`.

get_soil_texture_class

Infer soil texture class from particle fractions

Description

Given at least two of sand, silt, and clay percentages, this function infers the missing fraction and returns the corresponding USDA soil texture class (coarse classes).

Usage

```
get_soil_texture_class(sand = NA, silt = NA, clay = NA)
```

Arguments

sand	Numeric (0-100), percentage of sand.
silt	Numeric (0-100), percentage of silt.
clay	Numeric (0-100), percentage of clay.

Value

Character string giving the soil texture class (e.g. "CL", "L", "Sa", "SiL").

group_uncertainty_mc_trials

Build trial grouping table for MC uncertainty results

Description

Default grouping extracts parcel-year code patterns such as `2022_22526AL` from trial ids. If no such pattern is found, the last trial token without irrigation suffix is used.

Usage

```
group_uncertainty_mc_trials(x, group_fun = NULL)
```

Arguments

x	Result from `run_uncertainty_mc()` or `.rds` file path.
group_fun	Optional function taking a trial id and returning a group id.

Value

A data.frame with columns `trial_id` and `group_id`.

inject_datasets_soil *Inject inferred soil codes into OptirrigDatasets payloads*

Description

Inject inferred soil codes into OptirrigDatasets payloads

Usage

```
inject_datasets_soil(x, overwrite = FALSE, strict = TRUE)
```

Arguments

x	A trial payload or a named list of trial payloads.
overwrite	Logical. If TRUE, replace existing soil$soil.
strict	Logical. If TRUE, error when no code can be inferred.

Value

Same structure as x with filled soil$soil where possible.

msg *Automatically Format and Display Styled Messages*

Description

Formats and displays messages with custom styling using the cli package. The messages can include clickable file paths and are styled based on the type provided.

Usage

```
msg(text = "", ..., path = NULL, type = "T", verbose = TRUE)
```

Arguments

text	A character string containing the main message.
...	Additional text elements to append to the main message.
path	An optional character string specifying a file path that will be displayed as an italicized, clickable link.
type	A character specifying the message type. Examples: "S" (success), "W" (warning), "D" (danger), "I" (info), "T" (plain text).
verbose	A logical value indicating whether the message should be displayed. Defaults to TRUE.

Details

This function uses a custom cli theme to differentiate various message types, ensuring that success, warning, error, and informational messages are visually distinct.

---

null_coalescing_operator *Null-coalescing operator*

Description

Returns a unless it is NULL or NA, otherwise returns b.

Usage

```
a %||% b
```

Arguments

a	First value to check
b	Default value to return if a is NULL or NA

Value

Either a or b.

---

plot_uncertainty_mc *Plot MC global uncertainty (no OAT)*

Description

Plot MC global uncertainty (no OAT)

Usage

```
plot_uncertainty_mc(
  x,
  group_by = c("none", "trial_group"),
  grouping = NULL,
  group_map = NULL,
  group_fun = NULL,
  show_quantile_lines = TRUE,
  show_median_line = TRUE,
  x_axis = c("Date", "TT_sowing"),
  tt_step = 20,
  show = TRUE,
```

```
  save = FALSE,
  path = NULL,
  width = 16,
  height = 14,
  dpi = 300
)
```

Arguments

x	Result from run_uncertainty_mc(), output of prepare_uncertainty_mc_plot_data(), or .rds file path.
group_by	Either "none" (default) or "trial_group" to build one recap figure per parcel-year group directly from this function.
grouping	Optional common aggregation object. Supported forms: • a data.frame with columns trial_id and group_id • a list with map (or group_map) plus optional trial_col, group_col • a function f(trial_id) -> group_id If NULL, fallback uses group_map, then group_fun, then default parser.
group_map	Optional data.frame with trial_id and group_id used when group_by = "trial_group".
group_fun	Optional function used to derive group ids when group_by = "trial_group" and group_map is NULL.
show_quantile_lines	Logical; if TRUE, draw q05/q25/q75/q95 lines.
show_median_line	Logical; if TRUE, draw median line.
x_axis	Axis used for daily uncertainty ribbons: "Date" or "TT_sowing".
tt_step	Thermal-time grid step used when x_axis = "TT_sowing".
show	Logical. If TRUE, print the plot.
save	Logical. If TRUE, save the composed plot to path.
path	Optional output path. If NULL and save=TRUE, a default path is created in current working directory.
width, height, dpi	Export options passed to ggplot2::ggsave().

Value

If group_by = "none", invisibly returns a list with plot, panels, and path. If group_by = "trial_group", returns a named list of per-group plot results.

plot_uncertainty_mc_by_group *Plot MC uncertainty by parcel-year group (one recap per group)*

Description

Plot MC uncertainty by parcel-year group (one recap per group)

Usage

```
plot_uncertainty_mc_by_group(
  x,
  grouping = NULL,
  group_map = NULL,
  group_fun = NULL,
  show_quantile_lines = TRUE,
  show_median_line = TRUE,
  x_axis = c("Date", "TT_sowing"),
  tt_step = 20,
  show = TRUE,
  save = FALSE,
  path_dir = NULL,
  file_prefix = "uncertainty_mc",
  width = 16,
  height = 14,
  dpi = 300
)
```

Arguments

x	Result from run_uncertainty_mc() or .rds file path.
grouping	Optional common aggregation object. Supported forms: • a data.frame with columns trial_id and group_id • a list with map (or group_map) plus optional trial_col, group_col • a function f(trial_id) -> group_id If NULL, fallback uses group_map, then group_fun, then default parser.
group_map	Optional data.frame with trial_id and group_id.
group_fun	Optional function used when group_map is NULL.
show_quantile_lines	Logical; if TRUE, draw q05/q25/q75/q95 lines.
show_median_line	Logical; if TRUE, draw median line.
x_axis	Axis used for daily uncertainty ribbons: "Date" or "TT_sowing".
tt_step	Thermal-time grid step used when x_axis = "TT_sowing".

show	Logical. If TRUE, print each group plot.
save	Logical. If TRUE, save one file per group.
path_dir	Directory for exported plots when save = TRUE.
file_prefix	Prefix for exported file names.
width, height, dpi	Export options passed to ggplot2::ggsave().

Value

Named list of results returned by plot_uncertainty_mc(), one per group.

`prepare_uncertainty_mc_plot_data` *Prepare MC uncertainty data for plotting (global MC only)*

Description

Prepare MC uncertainty data for plotting (global MC only)

Usage

```
prepare_uncertainty_mc_plot_data(
  x,
  daily_vars = c("I", "Res", "Sw", "LAI", "B", "Y_sugar"),
  plant_vars = c("LAI", "B", "Y_sugar"),
  seasonal_vars = c("sumIRR", "Y_ratio", "Wprod", "Iprod"),
  x_axis = c("Date", "TT_sowing"),
  tt_step = 20
)
```

Arguments

x	Object returned by run_uncertainty_mc() or path to an .rds file.
daily_vars	Character vector of daily variables to plot.
plant_vars	Character vector of plant variables to plot.
seasonal_vars	Character vector of seasonal indicators to plot.
x_axis	Axis used for daily uncertainty ribbons: "Date" or "TT_sowing".
tt_step	Thermal-time grid step used when x_axis = "TT_sowing".

Value

A list with standardized plotting tables and labels.

`randomize_rainfall` *Dry out a precipitation series randomly and display a comparative barplot.*

Description

Dry out a precipitation series randomly and display a comparative barplot.

Usage

```
randomize_rainfall(
  precip,
  prob_remove = 0.2,
  min_factor = 0.3,
  max_factor = 0.9,
  col_modified = "red",
  seed = NULL,
  date_sowing = NULL,
  date_harvest = NULL,
  dates = NULL
)
```

Arguments

`precip`	Numeric. Precipitation series (>=0).
`prob_remove`	Probability of completely removing a rainy day.
`min_factor`	Minimum multiplicative factor (reduced days).
`max_factor`	Maximum multiplicative factor (reduced days).
`col_modified`	Color of the modified series (examples: "red", "blue").
`seed`	Optional. For reproducibility.
`date_sowing`	Date (or index) marking the beginning of the crop window used to report cumulative rainfall.
`date_harvest`	Date (or index) marking the end of the crop window used to report cumulative rainfall.
`dates`	Optional vector of dates aligned with `precip`. Required when `date_sowing` / `date_harvest` are supplied as dates or characters.

Value

A modified precipitation vector (same length as original).

resize_logo *Resize Image to Multiple Widths*

Description

Reads an image from a specified file path, resizes it to multiple specified widths while preserving the aspect ratio, and saves the resized images with filenames that include the respective width.

Usage

```
resize_logo(filePath, widths)
```

Arguments

filePath	A character string specifying the path to the source image file. Supported formats include PNG, JPG, etc.
widths	A numeric vector specifying the target widths (in pixels) for the resized images.

Details

The function reads the original image, computes the corresponding height for each specified width to maintain the original aspect ratio, and then resizes the image accordingly. The resized images are saved with filenames that include the width as a suffix (e.g., image_300px.png).

The magick package is used to handle image manipulation, ensuring that the resizing operation preserves the quality of the image. The resized images are saved in the PNG format by default.

Value

A character vector containing the file paths of the resized images.

A character vector containing the file paths to the resized images.

run_calibration_deoptim *Run Calibration with* DEoptim

Description

Run Calibration with DEoptim

Usage

```
run_calibration_deoptim(fn, inputs, lower, upper, control, options)
```

Arguments

fn	Objective function.
inputs	Inputs passed to fn.
lower	Lower bounds.
upper	Upper bounds.
control	Optimizer controls.
options	Options passed to fn.

Value

A calibration object.

---

run_calibration_neldermead

Run Calibration with neldermead

Description

Run Calibration with neldermead

Usage

```
run_calibration_neldermead(fn, inputs, lower, upper, control, options)
```

Arguments

fn	Objective function.
inputs	Inputs passed to fn.
lower	Lower bounds.
upper	Upper bounds.
control	Optimizer controls.
options	Options passed to fn.

Value

A calibration object.

run_calibration_optim *Run Calibration with* optim

Description

Run Calibration with optim

Usage

```
run_calibration_optim(fn, inputs, lower, upper, control, options)
```

Arguments

fn	Objective function.
inputs	Inputs passed to fn.
lower	Lower bounds.
upper	Upper bounds.
control	Optimizer controls.
options	Options passed to fn.

Value

A calibration object.

run_calibration_screen *Run Calibration with parameter screening*

Description

Run Calibration with parameter screening

Usage

```
run_calibration_screen(fn, inputs, lower, upper, control, options)
```

Arguments

fn	Objective function.
inputs	Inputs passed to fn.
lower	Lower bounds.
upper	Upper bounds.
control	Screening controls.
options	Options passed to fn.

Value

A calibration object.

`run_model_calibration` *Objective Wrapper for Calibration*

Description

Simple objective wrapper for calibration workflows. It takes calibration `inputs` (typically specs), selected `params`, and `options`, then:

1. builds model inputs,
2. runs `OptirrigCORE::run_model()`,
3. runs `run_model_eval()`.

Usage

```
run_model_calibration(inputs, params, options = list())
```

Arguments

`inputs`	Calibration inputs. Can be: • a list of specs (e.g. from `calib_build_specs()`).
`params`	Named numeric vector of selected parameters.
`options`	Named list of options.

Value

A list with:

- `method`: criterion label returned by `run_model_eval()`,
- `score`: scalar objective score,
- `timings`: run/eval/total elapsed seconds.

run_model_eval *Compute calibration criterion from pooled Obs/Sim across runs*

Description

The function extracts observations from `inputs[[i]]$obs`, extracts matching simulated variables from `outputs[[i]]`, joins Obs/Sim by `run_id` + `Date` + `var`, computes one criterion per variable on pooled values across runs, then aggregates variable scores to a final score.

Usage

```
run_model_eval(
  inputs,
  outputs,
  method = c("nrmse", "rmse", "prmse", "mse", "mae", "mre", "d"),
  FUN = NULL,
  var_weights = NULL,
  min_obs_n = 3L,
  single_obs_weight = 1
)
```

Arguments

`inputs`	List of run inputs. Each run must contain at least `run$run_id` and `obs`.
`outputs`	List of model outputs.
`method`	Criterion method used when `FUN` is `NULL`. Supported values: `"nrmse"`, `"rmse"`, `"prmse"`, `"mse"`, `"mae"`, `"mre"`, `"d"`. `"prmse"` is an alias of `"rmse"`.
`FUN`	Optional custom criterion function `(obs, sim) -> numeric(1)`.
`var_weights`	Optional named weights by variable, e.g. `list(LAI = 0.3, Y_sugar = 0.7)`. If provided, unspecified variables default to weight 1. Advanced format is also supported for dynamic time-weighting: `list(LAI = c(1, 0.4), Y_sugar = c(0.4, 1))`, interpreted as linear weights from sowing date to run end (`start -> end`) for each variable.
`min_obs_n`	Minimum number of observed values required to keep one `run_id` + `var` series.
`single_obs_weight`	Relative point weight applied to one `run_id` + `var` series when it has exactly one observed value. Series with more than one observed value keep weight `1`.

Value

A list with:

- `method`
- `score`
- `score_by_var`

- score_tbl
- obs_sim

---

run_model_eval_ed *Compute ED-based calibration criterion across trial pairs*

Description

This objective compares distances between trial trajectories instead of raw Obs/Sim points. For each variable, it computes one Euclidean-distance summary per site and trial pair on simulated values and on observed values, then measures the gap between both summaries.

Usage

```
run_model_eval_ed(
  inputs,
  outputs,
  vars = "Y_sugar",
  ed_method = c("classic", "symmetric", "symetric"),
  ed_stat = c("mean", "median", "sd", "max", "min"),
  loss = c("rmse", "mae", "bias_abs"),
  trial_levels = c("Ndc", "Irri", "ArrT", "ArrP", "IrriMn", "DclT", "Tem"),
  var_weights = NULL,
  min_obs_n = 3L
)
```

Arguments

inputs	List of run inputs.
outputs	List of model outputs.
vars	Variables used to build the ED objective.
ed_method	ED method: "classic" or "symmetric".
ed_stat	Statistic used to aggregate ED within each trial pair.
loss	Loss used to compare simulated vs observed ED: "rmse", "mae", or "bias_abs".
trial_levels	Scenario suffix levels used to parse run_id.
var_weights	Optional named weights by variable.
min_obs_n	Minimum number of observed values required to keep one run_id + var series.

Value

A list with method, score, score_by_var, obs_sim, ed_summary, and ed_gap.

run_uncertainty_mc *Run Monte-Carlo uncertainty from calibration specs*

Description

Run Monte-Carlo uncertainty from calibration specs

Usage

```
run_uncertainty_mc(
  x,
  funs = NULL,
  n_draws = 50L,
  sampling = c("normal", "lhs", "uniform"),
  quantile_probs = c(0.05, 0.5, 0.95),
  forced_draws = NULL,
  seed = NULL,
  plan = "multicore",
  workers = NULL,
 output_vars = c("Date", "P", "I1", "RU", "RUmax", "stress_ETR", "B", "Bp", "Y", "Yp",
    "Y_sugar", "Yp_sugar", "LAI", "W_eff"),
  inputs_path = NULL,
  log = FALSE,
  verbose = TRUE,
  return_outputs = FALSE,
  output_path = NULL,
  rds_name = "uncertainty_mc.rds",
  force = FALSE,
  grouping = NULL
)
```

Arguments

x	A `create_spec()` object or a list of specs.
funs	Optional named list of objective functions.
n_draws	Number of Monte-Carlo draws.
sampling	Sampling method: `"normal"`, `"lhs"` or `"uniform"`.
quantile_probs	Optional vector of probabilities in `0, 1`.
forced_draws	Optional forced parameter draws to append to the Monte-Carlo sample. Can be a named numeric vector, a data frame, or a list of such objects. Required columns/names must match the calibrated parameter names.
seed	Optional random seed.
plan	Execution plan forwarded to `run_model()`.
workers	Optional number of workers forwarded to `run_model()`.

output_vars	Character vector of model variables requested from run_model().
inputs_path	Optional path to project inputs/.
log	Logical. Forwarded to run_model().
verbose	Logical. Print workflow progress.
return_outputs	Logical. If TRUE, keep raw run_model() outputs.
output_path	Optional directory for RDS cache.
rds_name	File name used in output_path.
force	Logical. If FALSE and RDS exists, load cache.
grouping	Optional grouping object stored in the result.

Value

A list with draws, run map, daily summaries, seasonal metrics, optional quantiles, and optional raw outputs.

`select_screen_representatives` *Select representative screening candidates*

Description

Cluster the best screening candidates below a score threshold and keep one low-score representative per cluster.

Usage

```
select_screen_representatives(
  screen_res,
  param_cols,
  score_gate = 0.2,
  n_representants = 5L
)
```

Arguments

screen_res	Screening result object containing optim$results.
param_cols	Character vector of parameter column names used to compute distances between candidates.
score_gate	Maximum score retained in the candidate pool.
n_representants	Number of representative clusters to retain.

Value

A list containing the filtered candidate pool, selected representatives, a representative parameter table, a cluster summary, and the number of clusters used.

`stylifier` *Apply Code Styling and Detect Encoding Issues*

Description

This function applies stylistic formatting to R and Rmd files in specified directories using the styler package. It also detects non-ASCII (encoding) issues inside function definitions and forces UTF-8 encoding on each file.

Usage

```
stylifier(directory = c("R", "vignettes", "reports", "inst"), recursive = TRUE)
```

Arguments

`directory`	A character vector of directory names to search for files ending with `.R` or `.Rmd`. Defaults to c("R", "vignettes", "reports", "inst").
`recursive`	A logical value indicating whether to search directories recursively. Defaults to TRUE.

Details

Files are scanned for non-ASCII characters only within function bodies. Upon finding any issues, the function logs detailed information before applying style formatting using styler::style_file.

Index