

Package: kwb.raindrop (via r-universe)

October 6, 2026

Title R Package for Optimisation Simulations for Rainwater Management
Simulations Performed with Calculation Engine Provided by
Tandler

Version 0.1.0

Description R Package for Optimisation Simulations for Rainwater
Management Simulations Performed With Calculation Engine
Provided by Tandler.

License MIT + file LICENSE

URL <https://github.com/KWB-R/kwb.raindrop>

BugReports <https://github.com/KWB-R/kwb.raindrop/issues>

Imports dplyr, forcats, fs, future, future.apply, ggplot2, hdf5r,
kwb.event, kwb.utils, lubridate, magrittr, parallel, progressr,
purrr, rlang, stringr, tibble, tidyr

Suggests covr, DT, htmlwidgets, knitr, plotly, readr, rmarkdown,
testthat (>= 3.0.0), writexl

VignetteBuilder knitr

Config/testthat/edition 3

Remotes github::kwb-r/kwb.event, github::kwb-r/kwb.utils

Encoding UTF-8

Roxygen list(markdown = TRUE)

Config/roxygen2/version 8.1.0

Config/pak/sysreqs cmake make libhdf5-dev libicu-dev libuv1-dev

Repository <https://kwb-r.r-universe.dev>

Date/Publication 2026-08-07 13:06:57 UTC

RemoteUrl <https://github.com/KWB-R/kwb.raindrop>

RemoteRef HEAD

RemoteSha 7fe792f52c11ce3c5bc41260c65b83cb29fec247

Contents

add_overflow_events_and_waterbalance	3
compute_costs	4
cost_rates_caption	6
default_canonical_wb_variables	7
default_cost_rates	7
default_param_labels	8
default_storage_spec	9
default_storage_types	9
download_engine	10
find_min_feasible	11
find_single_param_variations	13
get_simulation_results_all	14
get_simulation_results_optim	15
get_simulation_results_optim_parallel	16
h5_ensure_dataset	17
h5_ensure_datasets_from_values	18
h5_read_values	19
h5_validate_write	19
h5_write_values	20
list_h5_datasets	21
make_swale_runner	22
optimise_swale_design	23
optimise_swale_design_simultaneous	25
plot_cost_overflow_boxplot	29
plot_cost_vs_evaporation	32
plot_cost_vs_overflow_volume	34
plot_hpond_vs_ref	36
plot_main_effects	37
plot_valid_design_space	38
plot_wb_tradeoff_overflows	40
plotly_add_caption	42
plotly_split_legend	43
read_hdf5_connections	44
read_hdf5_scalars	46
read_hdf5_timeseries	47
read_raindrop_errors	48
read_site_timeseries	49
run_model	50
run_scenarios	51
sickerbox_level_presets	52
stack_levels	53

Index

add_overflow_events_and_waterbalance

Add overflow-event metrics and water-balance shares (percent) to simulation results

Description

Computes (i) overflow event statistics from the element outflow time series and (ii) water-balance components as percent shares for both element and connected_area per scenario.

Usage

```
add_overflow_events_and_waterbalance(
  simulation_results,
  event_separation_hours = 4,
  canonical_variables = NULL
)
```

Arguments

simulation_results

Named list of scenario results. Each entry is expected to contain:

- element\$water_balance with columns variable, value
- connected_area\$water_balance with columns variable, value
- element\$rates with columns time, variable, value

event_separation_hours

Numeric. Minimum time between two overflow events (in hours). Defaults to 4.

canonical_variables

Optional character() vector of water-balance variable names (without the element. / connectedarea. prefix and without the trailing _), e.g. default_canonical_wb_variables(). This is a **per-scenario** fallback: for any scenario whose wb_element and wb_connectedarea are both empty after the regular pivot / mirror logic (including scenarios that are entirely NULL), the function attaches element.<var>_ and connectedarea.<var>_ NA-filled stub columns built from this list. This guarantees the output tibble keeps the expected water-balance column structure even when no scenario contributes real data — dplyr::bind_rows() would otherwise drop columns that no row supplies. Defaults to NULL (no canonical fallback).

Details

Water-balance percentages are computed with sign preserved (value / denom):

- **element** denominator: WB_Regen + abs(WB_Oberflaechenablauf_Verschaltungen)
- **connected_area** denominator: WB_Regen (fallback to abs(WB_Oberflaechenablauf_Verschaltungen) if WB_Regen is NA or 0)

Overflow events are derived from positive `Oberflaechenablauf_Ueberlauf` values using `kwb.event::hsEvents()`. Assuming the overflow rate is in mm/h, event sums (in mm) are obtained by integrating each sample over its **local** time step (`time[i+1] - time[i]`; the last sample inherits the previous step). A warning is emitted if the time step is non-uniform, and `sum_overflows` is returned as NA if it cannot be determined (single sample).

Missing components are tolerated: scenarios whose entry in `simulation_results` is NULL (or which lack any of `element`, `element$water_balance`, `connected_area`, `connected_area$water_balance` or `element$rates`) still produce a row of the output tibble. The four "headline" columns (`s_name`, `n_overflows`, `median_duration_overflows_hours`, `sum_overflows`) are always present and filled with NA where they cannot be computed.

For the `element.*_` and `connectedarea.*_` water-balance columns: if one side is missing in a given scenario but the other side has data, a stub of NA-filled columns is fabricated for the missing side by mirroring the populated side's variable names. This preserves the table's column structure when, for example, every run disables roof ET (so the engine skips writing `Dach.h5` and every scenario has `connected_area = NULL`): the `connectedarea.*_` columns are kept and filled with NA, instead of being dropped from the output entirely. The mirror is a best-effort hint for the user, not a guarantee that the names match what a populated `connected_area` would have produced.

Value

A tibble with one row per scenario containing:

- `s_name`
- `n_overflows`
- `median_duration_overflows_hours`
- `sum_overflows`
- water-balance percentage columns for `element.*_` and `connectedarea.*_`

Examples

```
## Not run:
out <- add_overflow_events_and_waterbalance(simulation_results)
out <- add_overflow_events_and_waterbalance(
  simulation_results,
  event_separation_hours = 6
)

## End(Not run)
```

compute_costs

Compute construction costs for an infiltration-swale parameter grid

Description

Given a parameter grid that drives the simulation, attach a per-scenario breakdown of construction costs plus a `cost_total` column suitable for filtering / sorting in the results datatable.

Usage

```
compute_costs(
  param_grid,
  storage_type = c("infiltration_box", "gravel_trench"),
  cost_rates = default_cost_rates()
)
```

Arguments

param_grid	data.frame / tibble — must contain the four geometry columns listed above. Other columns are passed through unchanged.
storage_type	character(1) — "infiltration_box" (default; ~95% porosity, the Austrian "Sickerbox") or "gravel_trench" (~30% porosity, the Austrian "Schotterrigol"). Only used when param_grid does not have a storage_type column.
cost_rates	list — unit costs as returned by default_cost_rates() . Override individual entries to run cost sensitivity analyses.

Details

Required columns in param_grid:

- mulde_area — element footprint area in m²
- mulde_height — surface depression depth in **mm**
- filter_height — soil-filter layer thickness in **mm**
- storage_height — storage layer thickness in **mm**

Optional column storage_type (character: "infiltration_box" or "gravel_trench") selects the per-scenario storage rate. If absent, the value of the storage_type argument is used for every row.

Cost formulas (Leimgruber, 2026-03-27):

```
cost_excavation = mulde_area * (mulde_h + filter_h + storage_h)/1000 * 70
cost_profiling  = mulde_area * 10
cost_filter     = mulde_area * filter_h/1000 * 200
cost_storage    = mulde_area * storage_h/1000 * <350 | 50>
cost_total      = sum of the four above
```

Value

The input param_grid with the columns storage_type, cost_excavation, cost_profiling, cost_filter, cost_storage, cost_total (all in EUR) appended.

Examples

```
grid <- tibble::tibble(
  scenario_name = c("s00001", "s00002"),
  mulde_area = c(50, 50),
  mulde_height = c(300, 300),
```

```

    filter_height = c(300, 300),
    storage_height = c(600, 600)
  )

  # Infiltration box (default)
  compute_costs(grid)

  # Gravel trench
  compute_costs(grid, storage_type = "gravel_trench")

  # Per-scenario storage type
  compute_costs(dplyr::mutate(grid,
                              storage_type = c("infiltration_box",
                                                "gravel_trench")))

```

cost_rates_caption	<i>Caption line naming the unit-cost rates behind the cost plots</i>
--------------------	--

Description

Formats the unit-cost rates (EUR per m² / m³, see [default_cost_rates\(\)](#)) as a single-line caption for the cost plots, so every figure names the rates its EUR values were computed with. Used as the default caption of [plot_cost_vs_overflow_volume\(\)](#), [plot_cost_vs_evaporation\(\)](#) and [plot_cost_overflow_boxplot\(\)](#) (rendered by ggplot at the bottom of the PDFs) and passed to [plotly_add_caption\(\)](#) for the interactive HTMLs (plotly::ggplotly() drops ggplot captions).

Usage

```
cost_rates_caption(lang = c("de", "en"), cost_rates = default_cost_rates())
```

Arguments

lang	Character. "de" or "en".
cost_rates	list of unit costs as returned by default_cost_rates() .

Details

If the costs were computed with non-default rates, pass the same cost_rates list here so the caption matches the numbers.

Value

character(1), a single line.

Examples

```
cost_rates_caption("de")
```

```
default_canonical_wb_variables
```

Canonical water-balance variable names emitted by the Tandler engine

Description

Returns the list of water-balance scalar names the Tandler "Regenwasserbewirtschaftung" engine writes into the Wasserbilanz group of each result HDF5 (without the element. / connectedarea. prefix and without the trailing _ that [add_overflow_events_and_waterbalance\(\)](#) appends when building the output columns). Pass this to that function's canonical_variables argument to keep the expected water-balance column structure visible in the rendered datatable even when every scenario in a batch is NULL (e.g. the engine returns Status 1 for every input and writes no result HDF5).

Usage

```
default_canonical_wb_variables()
```

Value

A character() vector of variable names.

Source

Values observed in the May 2026 432-row Wien sweep (Michael Rustler) and referenced explicitly by [plot_wb_tradeoff_overflows\(\)](#).

Examples

```
default_canonical_wb_variables()
```

```
default_cost_rates
```

Default unit-cost rates for infiltration-swale installations

Description

Returns the default Austrian unit-cost rates used by [compute_costs\(\)](#). Rates were provided by Johannes Leimgruber (OeStaP) on 2026-03-27 for the RAINDROP cost-optimisation work.

Usage

```
default_cost_rates()
```

Value

A named list of rates in EUR per m² or m³: excavation_eur_per_m3, profiling_eur_per_m2, filter_eur_per_m3, infiltration_box_eur_per_m3, gravel_trench_eur_per_m3.

Source rates (Leimgruber, 2026-03-27, "Kostenansätze Optimierung")

- Aushub (excavation, incl. loading + transport): 70 EUR/m³
- Profilierung und Begrünung (profiling + greening): 10 EUR/m²
- Bodenfilter (soil filter, incl. installation): 200 EUR/m³
- Sickerbox (infiltration box, incl. installation): 350 EUR/m³
- Schotterrigol (gravel trench, incl. installation): 50 EUR/m³

default_param_labels *German / English labels for optimisation parameter-grid columns*

Description

Maps the raw param_grid column names produced by the case-study workflows to human-readable, unit-carrying labels. Used to translate the "varying parameters" block in the interactive tooltips of `plot_cost_vs_overflow_volume()` and `plot_cost_overflow_boxplot()`, so a hovered point shows e.g. Muldenflaeche [m2]=125 instead of the raw mulde_area=125.

Usage

```
default_param_labels(lang = c("de", "en"))
```

Arguments

lang Character. "de" or "en".

Details

Unknown columns fall back to their raw name, so a grid gaining a new column still renders (just untranslated). Override individual entries or pass your own named vector via the param_labels argument of the plot functions.

Value

A named character vector: names are param_grid column names, values are the display labels.

Examples

```
default_param_labels("de")[["mulde_area"]]
default_param_labels("en")[["storage_height"]]
```

default_storage_spec *Default storage specification for the swale-design optimiser*

Description

Storage-layer search space per storage type: the infiltration box uses discrete stack levels (default: the brute-force grid levels), the gravel trench is continuous with bounds coupled to the box level range by coupling_factor (default 3, approximating the usable-porosity ratio 0.95 / 0.3).

Usage

```
default_storage_spec(
  levels = sickerbox_level_presets()$brute_force,
  coupling_factor = 3,
  gravel_tol = 25
)
```

Arguments

levels	Numeric vector of infiltration-box stack heights in mm.
coupling_factor	Factor between gravel-trench bounds and the box level range.
gravel_tol	Bisection tolerance for the continuous gravel-trench height in mm.

Details

Each entry also carries the **usable porosity** of the storage layer (box 0.95, trench 0.3, matching [default_storage_types\(\)](#)). The bisection optimiser uses it to *derive* its search order from the cost rates (cost per mm of storage capacity); without a porosity entry it falls back to the default-rate hierarchy (smallest storage level first).

Value

Named list with entries infiltration_box (with levels and porosity) and gravel_trench (with bounds, tol and porosity).

default_storage_types *Default storage-type soil presets (Speicher layer)*

Description

Soil parameters of the storage (2nd Bodenschichtung) layer per storage type, as used by the work-flow vignettes: infiltration box ("Sickerbox", thetaS 0.95) and gravel trench ("Schotterrigol", thetaS 0.3).

Usage

```
default_storage_types()
```

Value

Named list (per storage type) of lists with Startwerte_theta_ActualSoilMoisture, thetaWP_MoistureAtWiltingPoint, thetaFC_MoistureAtFieldCapacity, thetaS_MoistureAtSaturation.

download_engine	<i>Download the Tandler "Regenwasserbewirtschaftung" calculation engine</i>
-----------------	---

Description

Fetches the Windows executable from the corresponding GitHub Release of the companion KWB-R/kwb.raindrop.binaries repository and caches it under a per-version sub-directory of the user cache. Subsequent calls are no-ops once the file is present.

Usage

```
download_engine(  
  version = default_engine_version(),  
  cache_dir = tools::R_user_dir("kwb.raindrop", "cache"),  
  force = FALSE  
)
```

Arguments

version	character(1) Engine release version, matching the part after engine- in the KWB-R/kwb.raindrop.binaries Release tag. Defaults to the package's pinned version.
cache_dir	character(1) Directory where engine binaries are cached. A sub-directory named after version is used so multiple versions can coexist. Defaults to <code>tools::R_user_dir("kwb.raindrop", "cache")</code> .
force	logical(1) If TRUE, re-download even if the executable already exists in the cache. Default: FALSE.

Details

Releases in KWB-R/kwb.raindrop.binaries follow the tag scheme engine-<version> and contain a single asset named Regenwasserbewirtschaftung.exe (the version is encoded in the tag, not in the filename, so multiple engine versions can coexist side-by-side in the cache).

The executable is a Windows binary; on non-Windows platforms the file is still downloaded but cannot be executed directly. The function does not validate the download with a checksum — pin a specific version to guarantee reproducibility.

Value

character(1) — absolute path to the cached .exe.

Examples

```
## Not run:
exe <- download_engine()
exe <- download_engine("2026-02-24")

## End(Not run)
```

find_min_feasible	<i>Smallest feasible parameter value via bisection (monotone threshold search)</i>
-------------------	--

Description

Core building block of the swale-design optimiser: finds the smallest value of one design parameter for which the overflow target is met ($n_{\text{overflows}} \leq x_{\text{max}}$), assuming quasi-monotone feasibility (larger value = never more overflows; verified for the RAINDROP model in the monotonicity analysis, https://raindrop.kompetenz-wasser.io/optimisation/monotonicity_analysis/). Each evaluation halves the search interval, so $\text{ceiling}(\log_2(\text{range} / \text{tol}))$ evaluations suffice.

Usage

```
find_min_feasible(
  evaluate,
  x_max,
  lower = NULL,
  upper = NULL,
  tol = 1,
  levels = NULL,
  wobble = 1L,
  split_jitter = 0,
  volume_column = "overflow_volume_m3",
  verbose = FALSE
)
```

Arguments

evaluate	function(value) returning a list / one-row data.frame with at least $n_{\text{overflows}}$; if it also contains volume_column, the volume referee is active. Evaluations are memoised per value.
x_max	Feasibility target: feasible iff $n_{\text{overflows}} \leq x_{\text{max}}$.
lower, upper	Numeric search bounds (continuous mode).
tol	Resolution of the continuous search (same unit as the value).

levels	Sorted numeric vector of discrete candidate values (discrete mode, e.g. Sicker-box stack heights). If given, lower, upper and tol are ignored and a binary search over the levels is performed.
wobble	Maximum counting-artefact size tolerated by the edge guard (default 1, matching the observed +1 flips).
split_jitter	Numeric in $[0, 0.45]$, default 0. With 0 the interval is split exactly in half (deterministic). A positive value draws the split fraction uniformly from $0.5 \pm \text{split_jitter}$ – a Monte-Carlo of the <i>search path</i> : repeated runs with different seeds take different routes to the threshold and must agree within tol if the result is a property of the problem, not of the path.
volume_column	Name of the volume element in the evaluate result used by the volume referee (default "overflow_volume_m3").
verbose	Print one line per evaluation.

Details

Two safety rules from the monotonicity analysis are built in:

- **Edge guard:** if the upper bound is infeasible by no more than wobble events (the +1 counting artefact of the 4-h event separation), a descending ladder below the edge searches for a feasible anchor before the branch is declared infeasible.
- **Volume referee:** whenever `n_overflows` increases with the parameter (a counting flip), the overflow volume must have decreased; if the volume increased as well, a warning is emitted and `monotonicity_violation` is set (real non-monotonicity – never observed at the three validation sites).

Value

List with

value smallest feasible value, or NA if infeasible

n_overflows overflow count at value

status "ok", "at_lower_bound" (already feasible at the lower end – caller may widen the bracket) or "infeasible"

evaluations tibble of all evaluated values (value, n_overflows, volume), sorted by value

n_evaluations number of distinct evaluations

monotonicity_violation TRUE if the volume referee fired

Examples

```
# synthetic monotone step function: feasible from 137.4 m2 on
f <- function(v) list(n_overflows = if (v >= 137.4) 0L else 10L)
find_min_feasible(f, x_max = 0, lower = 25, upper = 200, tol = 2)$value
```

find_single_param_variations

Find scenarios that differ from a reference in exactly one parameter

Description

Identifies rows that differ from a reference in *exactly one* parameter. You may pass `param_cols` explicitly, or let the function infer them by excluding `id_col`, `exclude_cols`, and any columns matching `exclude_cols_regex` (default excludes result columns like `h_*`).

Usage

```
find_single_param_variations(
  data,
  ref_scenario = "s00001",
  id_col = "scenario_name",
  param_cols = NULL,
  exclude_cols = NULL,
  exclude_cols_regex = "^h_",
  tol = 1e-09,
  quiet = FALSE,
  include_reference = c("per_param", "none")
)
```

Arguments

<code>data</code>	A data frame with scenarios.
<code>ref_scenario</code>	Reference scenario ID (default "s00001").
<code>id_col</code>	Name of the scenario ID column (default "scenario_name").
<code>param_cols</code>	Optional character vector of parameter columns. If NULL, parameters are inferred.
<code>exclude_cols</code>	Character vector to exclude from parameter detection.
<code>exclude_cols_regex</code>	Regex to auto-exclude non-parameter columns (default "^h_").
<code>tol</code>	Numeric tolerance for numeric comparisons (default 1e-9).
<code>quiet</code>	Logical; print diagnostics if FALSE (default).
<code>include_reference</code>	"per_param" (default) to include one reference row per parameter with hits, or "none".

Value

A data frame with columns: `id_col`, `param_name`, `param_value`, `is_reference`. Attributes: `"param_cols"`, `"ref_row"`, `"diff_mat"`, `"n_diff"`.

```
get_simulation_results_all
```

Read Raindrop optimisation simulation results (all) from HDF5

Description

Reads per-run result files (HDF5) for both the *measure element* and the *connected area* results and returns a named list (one entry per simulation).

Usage

```
get_simulation_results_all(paths, path_list, simulation_names, debug = TRUE)
```

Arguments

<code>paths</code>	A list of path definitions. Used for messaging and expected to contain <code>file_results_hdf5_element</code> and <code>file_results_hdf5_flaeche</code> (file names). Note: within the loop, <code>paths</code> is overwritten by the resolved paths for the current simulation run.
<code>path_list</code>	A list passed to <code>kwb.utils::resolve()</code> to generate run-specific paths (must yield <code>path_results_hdf5_element</code> , <code>path_results_hdf5_flaeche</code> , and <code>dir_target_output</code>).
<code>simulation_names</code>	Character vector of simulation run identifiers (e.g. <code>c("s00001", "s00002")</code>).
<code>debug</code>	print debug messages (default: TRUE)

Details

For each simulation name (e.g. "s00001"), the function resolves the run directory via `kwb.utils::resolve(path_list, dir_target = s_name)` and then loads standard result groups from two HDF5 files:

- **element**: Metainfo, Raten, Wasserbilanz, Zustandsvariablen
- **connected_area**: Metainfo, Raten, Wasserbilanz, Zustandsvariablen

If either of the expected HDF5 files is missing for a run, the corresponding list entry will be NULL.

The function uses `hdf5r::H5File$new(..., mode = "r")` to open the files. The HDF5 handles are not explicitly closed; depending on your workflow, you may want to close them (see `hdf5r::H5File$close_all()` / `$close()`).

Value

A named list with one entry per `simulation_names`. Each entry is either NULL (missing files) or a nested list:

- element meta** Data.frame/list of scalar metadata (from Metainfo).
- rates** Time series table (from Raten).
- water_balance** Scalars table (from Wasserbilanz).
- states** Time series table (from Zustandsvariablen).
- connected_area** Same structure as **element**, read from the area HDF5.

See Also

[resolve](#), [read_hdf5_scalars](#), [read_hdf5_timeseries](#)

```
get_simulation_results_optim
```

Read Raindrop optimisation simulation results from HDF5

Description

Reads per-run result files (HDF5) for both the *measure element* and the *connected area* results and returns a named list (one entry per simulation).

Usage

```
get_simulation_results_optim(
  paths,
  path_list,
  simulation_names,
  debug = TRUE,
  lean = FALSE
)
```

Arguments

paths	A list of path definitions. Used for messaging and expected to contain <code>file_results_hdf5_element</code> and <code>file_results_hdf5_flaeche</code> (file names). Note: within the loop, paths is overwritten by the resolved paths for the current simulation run.
path_list	A list passed to <code>kwb.utils::resolve()</code> to generate run-specific paths (must yield <code>path_results_hdf5_element</code> , <code>path_results_hdf5_flaeche</code> , and <code>dir_target_output</code>).
simulation_names	Character vector of simulation run identifiers (e.g. <code>c("s00001", "s00002")</code>).
debug	print debug messages (default: TRUE)
lean	Logical. If TRUE, read only the fields consumed by add_overflow_events_and_waterbalance – <code>element\$rates</code> , <code>element\$water_balance</code> and <code>connected_area\$water_balance</code> – and leave <code>meta/states</code> (both sides) and <code>connected_area\$rates</code> as NULL. This keeps per-run memory and I/O minimal when each run is thinned to its optimisation row immediately instead of collecting every run's full results first. Defaults to FALSE (read everything).

Details

For each simulation name (e.g. "s00001"), the function resolves the run directory via `kwb.utils::resolve(path_list, dir_target = s_name)` and then loads standard result groups from two HDF5 files:

- **element:** Metainfo, Raten, Wasserbilanz, Zustandsvariablen
- **connected_area:** Metainfo, Raten, Wasserbilanz, Zustandsvariablen

If the **element** HDF5 is missing the entry is NULL; if only the **connected-area** HDF5 is missing, the entry is a list with the **element** side populated and **connected_area** = NULL. HDF5 handles are closed on exit via `on.exit(...$close_all())`.

The function uses `hdf5r::H5File$new(..., mode = "r")` to open the files and registers an `on.exit(...$close_all())` so handles are released even when the iteration body errors out.

Value

A named list with one entry per `simulation_names`. Each entry is either NULL (element HDF5 missing) or a nested list:

element meta Data.frame/list of scalar metadata (from `Metainfo`).

rates Time series table (from `Raten`).

water_balance Scalars table (from `Wasserbilanz`).

states Time series table (from `Zustandsvariablen`).

connected_area Same structure as **element**, read from the connected-area HDF5, or NULL if that file is missing for the run.

See Also

[resolve](#), [read_hdf5_scalars](#), [read_hdf5_timeseries](#)

`get_simulation_results_optim_parallel`

Read Raindrop optimisation simulation results from HDF5 (parallel via `future.apply` + `progress`)

Description

Parallel variant of `get_simulation_results_optim()` using `future.apply::future_lapply()` including optional progress reporting via **progressr**.

Usage

```
get_simulation_results_optim_parallel(
  paths,
  path_list,
  simulation_names,
  debug = TRUE,
  workers = NULL,
  show_progress = TRUE,
  future_seed = TRUE,
  progress_handler = progressr::handler_txtprogressbar
)
```

Arguments

paths	A list of path definitions. Used for messaging and expected to contain <code>file_results_hdf5_element</code> and <code>file_results_hdf5_flaeche</code> (file names). Note: within the loop, paths is overwritten by the resolved paths for the current simulation run.
path_list	A list passed to <code>kwb.utils::resolve()</code> to generate run-specific paths (must yield <code>path_results_hdf5_element</code> , <code>path_results_hdf5_flaeche</code> , and <code>dir_target_output</code>).
simulation_names	Character vector of simulation run identifiers (e.g. <code>c("s00001", "s00002")</code>).
debug	print debug messages (default: TRUE)
workers	Optional number of parallel workers. If not NULL, a temporary <code>future::multisession</code> plan is set.
show_progress	Logical (default TRUE).
future_seed	Passed to <code>future.apply::future_lapply()</code> .
progress_handler	A progressr handler function (default: <code>progressr::handler_txtprogressbar</code>). If NULL, no handler is set.

Value

Named list (see [get_simulation_results_optim](#)). As with the non-parallel sibling, the entry for a run is NULL only when the element HDF5 is missing; a missing connected-area HDF5 yields a partial result with `connected_area = NULL`.

h5_ensure_dataset	<i>Ensure that an HDF5 dataset exists (create if missing)</i>
-------------------	---

Description

Creates missing groups along the path and then creates the dataset. Designed for RAINDROP input files where missing scalar parameters can crash the model definition reader.

Usage

```
h5_ensure_dataset(h5, path, value, dtype = NULL, dims = NULL)
```

Arguments

h5	Open <code>hdf5r::H5File</code> (mode "a" or "r+").
path	Absolute dataset path (e.g. <code>"/Berechnungsparameter/Zeitschritt_Verschaltungen"</code>).
value	Initial value to write after creation (scalar, vector, matrix, or 2-col TS).
dtype	Optional. Either an H5T object or a string (<code>"double"</code> , <code>"integer"</code> , <code>"logical"</code> , <code>"character"</code>). If NULL, inferred from value.
dims	Optional integer vector. If NULL, inferred from value.

Value

Invisibly TRUE

h5_ensure_datasets_from_values

Ensure that all datasets referenced by a values list exist

Description

Creates missing groups and datasets for all names(values). Datasets are created with dtype/dims inferred from the corresponding value. No values are written here.

Usage

```
h5_ensure_datasets_from_values(
  h5,
  values,
  ts_cols = c("time", "value"),
  ts_layout = c("2xN", "Nx2"),
  ts_dtype = "double",
  strict = TRUE
)
```

Arguments

h5	Open hdf5r::H5File (mode "a" or "r+").
values	Named list; names are dataset paths (leading // allowed).
ts_cols	Character(2). Column names for TS (default time/value).
ts_layout	One of "2xN" or "Nx2". For RAINDROP typically "2xN" in HDF5.
ts_dtype	Either an H5T object or a string ("double", "integer", "logical", "float").
strict	Stop if creation fails.

Value

Invisibly, the character vector of created dataset paths.

h5_read_values	<i>Read values of all (or selected) datasets</i>
----------------	--

Description

Read values of all (or selected) datasets

Usage

```
h5_read_values(
  h5,
  paths = NULL,
  simplify_scalars = TRUE,
  timeseries_as_tibble = TRUE,
  ts_names = c("time", "value")
)
```

Arguments

h5	An open <code>hdf5r::H5File</code> .
paths	character vector; if <code>NULL</code> read all.
simplify_scalars	logical: simplify scalar datasets to length-1 atoms.
timeseries_as_tibble	logical: convert 2xN / Nx2 arrays to <code>tibble(time,value)</code> .
ts_names	character(2): names for time/value columns.

Value

Named list.

h5_validate_write	<i>Validate what would be written where (pre-flight check)</i>
-------------------	--

Description

Summarizes for each path:

- current dataset dims/maxdims (HDF5 order)
- intended value shape (R order)
- intended write shape (HDF5 order; for TS always 2xN)
- decision: `SKIP` / `WRITE` / `RESIZE` / `NEED_RECREATE` / `ERROR`

Usage

```
h5_validate_write(h5, values, ts_cols = c("time", "value"))
```

Arguments

h5	An open <code>hdf5r::H5File</code> .
values	Named list of values to write (names are HDF5 paths).
ts_cols	Character(2). Column names for TS (default time/value).

Details

Notes:

- 2-column data.frames/tibbles are treated as Nx2 in R and mapped to 2xN in HDF5.
- Many RAINDROP files store time series as HDF5 dims 2xN (appearing as Nx2 in R).

Value

tibble

h5_write_values	<i>Write (updated) values back into existing HDF5 datasets (robust for your hdf5r build)</i>
-----------------	--

Description

- Scalars: write with required args (`args=list()` or `args=1L` fallback).
- 2-col TS (data.frame/tibble): expects Nx2 in R; writes as 2xN in HDF5 (RAINDROP style), using explicit hyperslab `args=list(1:2, 1:N)` to avoid empty selections.
- If TS length changes and dataset `maxdims` blocks resize, the dataset is deleted via `parent$link_delete(name)` and recreated with `dims=2xN` (fixed), then written. (No `maxdims` argument used; compatible with your `create_dataset` signature.)

Usage

```
h5_write_values(
  h5,
  values,
  resize = TRUE,
  strict = TRUE,
  ts_cols = c("time", "value"),
  scalar_strategy = c("error", "first", "collapse"),
  collapse_sep = ";",
  ts_dtype = "double",
  verbose = FALSE
)
```

Arguments

h5	Open hdf5r::H5File.
values	Named list; names are dataset paths (leading // allowed).
resize	Logical. If TRUE tries set_extent() where possible.
strict	Logical. If TRUE stop on first error else warn and continue.
ts_cols	Character(2). Column names for TS (default time/value).
scalar_strategy	"error" "first" "collapse".
collapse_sep	Separator for collapse.
ts_dtype	Either an H5T object or one of "double", "float", "integer", "logical".
verbose	Logical.

Value

Invisibly, written paths.

list_h5_datasets	<i>List all datasets (recursive)</i>
------------------	--------------------------------------

Description

List all datasets (recursive)

Usage

```
list_h5_datasets(h5)
```

Arguments

h5	An open hdf5r::H5File.
----	------------------------

Value

A tibble with columns: path, obj_type, dims, maxdims.

make_swale_runner

Create a site-specific single-scenario runner for the optimiser

Description

Factors the `run_one()` function that was duplicated across the three workflow vignettes (Eisenstadt 2005, Wien, Bad Aussee) into one package-level closure factory. The returned function runs the RAINDROP engine for one parameter set and returns the thinned one-row optimisation result (overflow events + water balance), augmented with the input parameters and the overflow volume in m3.

Usage

```
make_swale_runner(
  path_list,
  timestep_hours = 0.1,
  timeseries_rain = NULL,
  timeseries_et = NULL,
  storage_types = default_storage_types(),
  event_separation_hours = 4,
  scenario_prefix = "o",
  cleanup = TRUE,
  debug = FALSE
)
```

Arguments

- | | |
|------------------------|--|
| path_list | Path definition list as used by the workflow vignettes (resolvable with <code>kwb.utils::resolve()</code>), must contain <code>path_base</code> , <code>path_exe</code> , <code>dir_input</code> , <code>dir_output</code> , <code>dir_target_output</code> , <code>path_target_input</code> , <code>path_results_hdf5_element</code> , <code>path_results_hdf5_flaeche</code> , <code>file_target</code>). |
| timestep_hours | Engine time step in hours (default 0.1). |
| timeseries_rain | Optional <code>data.frame</code> time/value (mm/h) written to <code>//Kurven/Regen</code> (the dataset must exist in <code>base.h5</code>); when given, the <code>//Kurven/Growth_1</code> and <code>//Kurven/Shading_1</code> end times are extended to the rain series end (skipped for templates without these curves) and <code>rain_factor</code> is ignored. Without <code>timeseries_rain</code> , a per-run <code>rain_factor != 1</code> requires <code>//Kurven/Regen</code> to exist as a time series in <code>base.h5</code> – a clear error is thrown otherwise. |
| timeseries_et | Optional <code>data.frame</code> time/value (mm/h) written to <code>//Kurven/ET0</code> . |
| storage_types | Soil presets of the storage layer per storage type, see <code>default_storage_types()</code> . |
| event_separation_hours | Event separation for overflow counting (default 4, as in the vignettes and the monotonicity analysis). |

scenario_prefix	Prefix for generated scenario names (default "o" -> o00001, o00002, ... – distinct from the grid runs s00001 ...).
cleanup	Delete each scenario's copied input file and output directory right after the thinned one-row result has been read (default TRUE). The optimisers only need that row; without the cleanup an optimisation run (hundreds of engine runs per task, each with its own copy of base.h5 plus all output HDF5s) fills the temp drive and the engine aborts with HDF5 errno = 28 ("No space left on device"). Set FALSE to keep all scenario files for debugging. Files of a <i>failed</i> run are always kept.
debug	Passed on to the engine/reader helpers.

Details

Site differences are covered by the arguments: Eisenstadt scales the rain curve shipped in base.h5 by rain_factor (leave timeseries_rain=NULL), Wien and Bad Aussee replace the rain and ET0 curves entirely (timeseries_rain / timeseries_et, values in mm/h as written by the vignettes).

On the first call the runner prepares a **site master file** once: base.h5 plus everything identical for every run (calculation settings, ET/rain time series). Each run then copies the master and writes only its ~15 small parameter datasets. Compared to the previous full read/rewrite of *all* datasets per run this removes the dominant per-run overhead of the optimisation searches (hundreds of runs; for Wien / Bad Aussee it skips rewriting the 15-year rain series on every single engine run).

Value

function(params) where params is a named list (or one-row data.frame) with mulde_area, mulde_height (mm), storage_type, storage_height (mm), connected_area (m2), filter_height (mm), filter_hydraulicconductivity (mm/h), bottom_hydraulicconductivity (mm/h) and optionally rain_factor (default 1) and lai (default 3.9). It returns a one-row tibble with the parameters, the scenario name and the optimisation metrics (n_overflows, sum_overflows in mm, overflow_volume_m3, water-balance shares).

See Also

[optimise_swale_design\(\)](#), [find_min_feasible\(\)](#)

optimise_swale_design *Find the cost-optimal swale design per overflow target*

Description

Coordinate-descent optimiser built from a single primitive ([find_min_feasible\(\)](#), bisection over one parameter): shrink the expensive lever first (mulde_area), then the cheap one (mulde_height); the storage layer starts at its smallest level and is only escalated when the area is stuck at its upper bound. The filter conductivity is expected to be fixed at the maximum via fixed (it is cost-free and dominant, see the monotonicity analysis, https://raindrop.kompetenz-wasser.io/optimisation/monotonicity_analysis/). Every engine run is cached, so the sweep over all x_targets and both storage types shares evaluations.

Usage

```

optimise_swale_design(
  run_fn,
  x_targets = 0:5,
  area_bounds = c(25, 200),
  area_tol = 2,
  height_bounds = c(100, 300),
  height_tol = 10,
  storage_spec = default_storage_spec(),
  fixed = list(connected_area = 1000, filter_height = 300, filter_hydraulicconductivity =
    360, bottom_hydraulicconductivity = 12),
  prior_results = NULL,
  split_jitter = 0,
  max_total_depth = NULL,
  cost_rates = default_cost_rates(),
  verbose = TRUE
)

```

Arguments

run_fn	function(params) running one scenario and returning at least n_overflows plus sum_overflows (mm) or overflow_volume_m3; typically created with make_swale_runner() . params is a named list of mulde_area, mulde_height, storage_type, storage_height plus everything in fixed.
x_targets	Integer vector of overflow targets (feasible $\Leftrightarrow$ n_overflows $\leq$ x), default 0:5.
area_bounds, area_tol	Search range (m2) and resolution for mulde_area. The found area sits up to one area_tol above the exact feasibility boundary (worst-case cost overshoot roughly area_tol x specific cost per m2, i.e. a few percent at the default 2 m2); thanks to the bisection, every <i>halving</i> of area_tol costs only one additional engine run per area search – the cheapest precision lever of this optimiser. It also shrinks the tolerance-artefact part of the final mulde_height trim (the values just below the height maximum at low x_targets).
height_bounds, height_tol	Search range (mm) and resolution for mulde_height.
storage_spec	Storage search space per type, see default_storage_spec() : discrete levels (infiltration box) or continuous bounds + tol (gravel trench).
fixed	Named list of parameters passed unchanged to run_fn (connected area, filter geometry, kf at maximum, ...). Must contain filter_height for the cost model.
prior_results	Optional data.frame with prior (grid) results in the workflow CSV schema, used as warm start (narrows the first area bracket to one grid step).
split_jitter	Passed to find_min_feasible() : 0 (default) = deterministic halving; > 0 randomises every bisection split point (Monte-Carlo of the search path – repeated runs with different seeds must agree within the search tolerances).

max_total_depth	Optional analytic depth constraint in mm: mulde_height + filter_height + storage_height <= max_total_depth (e.g. from DWA-A 138 groundwater clearance or cover requirements). Enforced without any simulation runs.
cost_rates	Unit costs, see default_cost_rates() .
verbose	Print one progress line per solved cell.

Details

The search order is derived from `cost_rates` via a specific-cost proxy (EUR per mm of storage capacity, capacity model $V \sim \text{area} * (\text{mulde_height} + \text{porosity} * \text{storage_height})$; the layer porosity comes from `storage_spec`, see [default_storage_spec\(\)](#): maximising `mulde_height` first is optimal for *any* rates under this cost model (it costs only excavation, while area pays every component), and the starting storage level is chosen as the cheapest level per mm of capacity – the smallest level under the default rates, a high level when e.g. the storage material is cheap. Without a porosity entry in `storage_spec` the legacy order (smallest level first) is used. The proxy is a first-order heuristic: it assumes capacity-additive levers and cannot rank parameters with nonlinear hydraulic effects (e.g. a variable filter conductivity) – for those, and as the assumption-free cross-check, use [optimise_swale_design_simultaneous\(\)](#), which carries `cost_rates` directly inside its objective.

Value

Tibble with one row per (storage type, x): the optimal design (`mulde_area`, `mulde_height`, `storage_height`), its metrics (`n_overflows`, `overflow_volume_m3`, `et_pct`), cost columns from [compute_costs\(\)](#), a status ("ok" or "infeasible_within_bounds"), `monotonicity_warning` (volume referee) and `n_runs_new` (fresh engine runs spent on this cell). All evaluated designs are attached as attribute "evaluations".

See Also

[optimise_swale_design_simultaneous\(\)](#) (alternative: all parameters at once via penalised Nelder-Mead, as an independent cross-check of the coordinate descent), [find_min_feasible\(\)](#), [make_swale_runner\(\)](#), [default_storage_spec\(\)](#)

optimise_swale_design_simultaneous

Find the cost-optimal swale design by simultaneous parameter search

Description

Alternative to the coordinate-descent optimiser ([optimise_swale_design\(\)](#), bisection per parameter): all design parameters – `mulde_area`, `mulde_height` and `storage_height` – are optimised **simultaneously**. Infeasible designs (`n_overflows > x`) are not excluded but penalised (any infeasible design is worse than any feasible one; the number of excess events grades the penalty, steering the search back towards feasibility), so the search moves freely through the full parameter space and can trade the parameters against each other in a single step – it does not rely on the per-parameter monotonicity that the bisection exploits.

Usage

```

optimise_swale_design_simultaneous(
  run_fn,
  x_targets = 0:5,
  area_bounds = c(25, 200),
  area_tol = 2,
  height_bounds = c(100, 300),
  height_tol = 10,
  storage_spec = default_storage_spec(),
  fixed = list(connected_area = 1000, filter_height = 300, filter_hydraulicconductivity =
    360, bottom_hydraulicconductivity = 12),
  prior_results = NULL,
  method = c("nelder_mead", "diff_evolution", "halton_search"),
  n_starts = 4,
  max_evals = 80,
  seed = 1,
  wobble = 1L,
  max_total_depth = NULL,
  cost_rates = default_cost_rates(),
  verbose = TRUE
)

```

Arguments

run_fn	function(params) running one scenario and returning at least n_overflows plus sum_overflows (mm) or overflow_volume_m3; typically created with make_swale_runner() . params is a named list of mulde_area, mulde_height, storage_type, storage_height plus everything in fixed.
x_targets	Integer vector of overflow targets (feasible :<=> n_overflows <= x), default 0:5.
area_bounds, area_tol	Search range (m2) and resolution for mulde_area.
height_bounds, height_tol	Search range (mm) and resolution for mulde_height.
storage_spec	Storage search space per type, see default_storage_spec() : discrete levels (infiltration box) or continuous bounds + tol (gravel trench).
fixed	Named list of parameters passed unchanged to run_fn (connected area, filter geometry, kf at maximum, ...). Must contain filter_height for the cost model.
prior_results	Optional data.frame with prior (grid) results in the workflow CSV schema, used as warm start (the cheapest feasible grid cell of the branch becomes the first start / seeds the population).
method	Search method, see Details: "nelder_mead" (default), "diff_evolution" or "halton_search" (the latter two mainly for comparison).
n_starts	Number of Nelder-Mead starts per (storage type, x) cell (default 4; only used by method = "nelder_mead"). Warm starts (prior, previous target) count towards this number, then the storage-ladder anchors, then the space-filling points.

max_evals	Soft cap on fresh engine runs per cell for the search phase: once reached, the search winds down (already cached designs remain free). The final multi-valley lattice polish adds its own runs on top (typically 20-50 per cell). Default 80 – thanks to the shared cache the later <code>x_targets</code> of a storage type stay cheaper.
seed	Integer seed of the internal deterministic generator used by method = "diff_evolution" (ignored by the other methods). R's global RNG state is not touched.
wobble	Maximum counting-artefact size tolerated at the upper corner (default 1, matching the +1 event-counting wobble): only if the maximal design overflows by more than wobble events is the cell declared infeasible without a search.
max_total_depth	Optional analytic depth constraint in mm: <code>mulde_height + filter_height + storage_height <= max_total_depth</code> (e.g. from DWA-A 138 groundwater clearance or cover requirements). Enforced without any simulation runs.
cost_rates	Unit costs, see <code>default_cost_rates()</code> .
verbose	Print one progress line per solved cell.

Details

Three search methods share this penalised objective (plus cache, tolerance snapping and final lattice polish) and differ only in how they propose candidates:

- "nelder_mead" (default): multistart Nelder-Mead simplex via `stats::optim()` – the recommended method.
- "diff_evolution": a compact differential evolution (DE/rand/1/bin, population 12, $F = 0.7$, $CR = 0.9$), included for comparison. Deterministic: it draws from an internal Park-Miller generator seeded with `seed` and leaves R's global RNG (`.Random.seed`) untouched.
- "halton_search": quasi-random space-filling sampling (Halton sequence, bases 2/3/5) – a deliberately simple baseline showing what the structured searches must beat.

Three ingredients keep the number of engine runs in check:

- **Snapping:** every candidate is snapped to the search tolerances (`area_tol`, `height_tol`, `storage_tol` / discrete levels) before evaluation, so the shared cache absorbs repeated visits and the sweep over all `x_targets` reuses runs.
- **Multistart:** `n_starts` deterministic starting points (prior warm start and the optimum of the previous overflow target first, then a *storage ladder* – one anchor start per storage level, smallest level first – then fixed space-filling points) guard against the simplex stalling on the plateaus that the snapping and the integer overflow count create, and make sure every storage level competes: along the feasibility boundary the cost valley is flat, so the cheapest (usually smallest) storage level is easily missed from a single start. Different starts take different search paths – the counterpart of `split_jitter` in the bisection optimiser. Every start receives an equal slice of the remaining `max_evals` budget (unused runs roll over).
- **Lattice polish:** an accelerated pattern descent (steps of 8/4/2/1 tolerances downwards, cheaper by construction) runs from the cheapest feasible design of every storage level visited – capped at the 6 cheapest levels, which only bites for the continuous gravel trench (the discrete box has at most a handful) – because the storage axis separates cost valleys that single coordinate steps cannot cross. Besides the per-axis down steps each round proposes a *boundary slide*

(area down with mulde_height at its maximum – the two-coordinate trade towards the cheap end of the feasibility boundary) and a *mulde_height floor probe* (at large x the overflow count saturates, so the whole lower height range can be feasible even when a +1 counting wobble blocks every single step). All are just evaluated candidates – no monotonicity assumption enters. The result is locally optimal on the tolerance lattice, whatever the search method delivered.

The discrete infiltration-box levels are mapped onto a continuous latent axis (each level owns an equal share of $[0, 1]$), the gravel trench is searched continuously. The filter conductivity is expected to be fixed at the maximum via fixed (cost-free and dominant, see the monotonicity analysis, https://raindrop.kompetenz-wasser.io/optimisation/monotonicity_analysis/). max_total_depth is enforced by construction (the mulde_height axis is compressed to the remaining depth), so no simulation runs are spent on depth-invalid designs.

Compared to `optimise_swale_design()` this needs considerably more engine runs per cell (typically 60-120 instead of ~15; search phase plus multi-valley polish) but serves as an independent cross-check: it can discover cheaper corners of the design space that coordinate descent would miss if the parameter interaction were stronger than the monotonicity analysis suggests.

Value

Tibble with one row per (storage type, x), same schema as `optimise_swale_design()` plus a method column: the optimal design (mulde_area, mulde_height, storage_height), its metrics (n_overflows, overflow_volume_m3, et_pct), cost columns from `compute_costs()`, a status ("ok" or "infeasible_within_bounds"), monotonicity_warning (TRUE if a strictly larger design produced more overflows *and* more overflow volume among the cell's evaluations) and n_runs_new (fresh engine runs spent on this cell). All evaluated designs are attached as attribute "evaluations".

See Also

`optimise_swale_design()` (coordinate descent / bisection), `make_swale_runner()`, `default_storage_spec()`

Examples

```
# synthetic monotone model: overflows fall with retention capacity
run <- function(params) {
  cap <- params$mulde_area *
    (params$mulde_height + 0.95 * params$storage_height)
  list(n_overflows = max(0, floor(3.6e5 / cap) - 3),
       sum_overflows = 800 * max(0, 3.6e5 / cap - 3))
}
opt <- optimise_swale_design_simultaneous(
  run, x_targets = 1,
  storage_spec = default_storage_spec()["infiltration_box"],
  verbose = FALSE
)
opt[, c("x", "mulde_area", "mulde_height", "storage_height", "cost_total")]
```

plot_cost_overflow_boxplot

Cost boxplot per overflow-event count, points sized by overflow volume

Description

Companion to [plot_cost_vs_overflow_volume\(\)](#). For every number of overflow events (x-axis) it draws a boxplot of the total construction cost (y-axis, EUR) across all scenarios with that count, overlaid with the individual scenarios as jittered points whose **size scales with** size_by – the overflow volume (m3, sum_overflows (mm) * mulde_area (m2) / 1000; the default) or the element evapotranspiration share (%). One best scenario per box is highlighted; best_by selects its objective – cheapest, smallest overflow volume, or highest evapotranspiration (cost as tie-breaker) – so the three variants trace three different frontier lines. label_best annotates the marker.

Usage

```
plot_cost_overflow_boxplot(
  simulation_results_optimisation,
  param_grid,
  x = 5,
  filter_n_gtx = FALSE,
  use_jitter = TRUE,
  jitter_width = 0.2,
  jitter_seed = 1L,
  max_point_size = 6,
  box_alpha = 0.35,
  point_alpha = 0.6,
  digits = 2L,
  digits_params = 4L,
  lang = c("de", "en"),
  param_labels = NULL,
  size_by = c("overflow_volume", "evapotranspiration"),
  best_by = c("min_cost", "min_overflow", "max_evapotranspiration"),
  y_var = c("cost_total", "cost_per_evap_pct"),
  facet_storage_type = FALSE,
  label_best = FALSE,
  title = NULL,
  lab_x = NULL,
  lab_y = NULL,
  caption = NULL,
  lab_size = NULL,
  mark_best = TRUE,
  connect_best = TRUE,
  legend_position = "right"
)
```

Arguments

simulation_results_optimisation	Data frame with the columns scenario_name, n_overflows, sum_overflows, mulde_area, element.WB_Evapotranspiration_, element.WB_InfiltrationNetto_, element.WB_Oberflaechenablauf_Ueberlauf_, cost_excavation, cost_profiling, cost_filter, cost_storage, cost_total, storage_type. Typically the joined output of <code>add_overflow_events_and_waterbalance()</code> and <code>compute_costs()</code> .
param_grid	Data frame with parameter grid. Must contain scenario_name.
x	Numeric threshold. Counts 0..x each get their own box; counts > x collapse into a single ">x" box.
filter_n_gtx	Logical. If TRUE, scenarios with n_overflows > x are dropped (removing the ">x" box) before plotting.
use_jitter	Logical. If TRUE, points are horizontally jittered.
jitter_width	Numeric. Horizontal jitter half-width.
jitter_seed	Integer. Seed for reproducible jitter.
max_point_size	Numeric. Point size for the largest (valid-region) size_by value; the smallest maps to a fixed minimum so no point vanishes.
box_alpha, point_alpha	Numeric in [0, 1]. Box-fill / point opacity.
digits	Integer. Rounding for numeric values in the tooltip.
digits_params	Integer. Rounding for parameter values in the tooltip.
lang	Character. Plot language: "de" or "en".
param_labels	Named character vector translating param_grid columns to tooltip labels, or NULL to use <code>default_param_labels()</code> for lang.
size_by	Character. Which variable drives the point area (and its legend): "overflow_volume" (default, m3) or "evapotranspiration" (the element evapotranspiration share in %, from element.WB_Evapotranspiration_ – larger points then mean <i>more</i> evapotranspiration, which is desirable).
best_by	Character. Objective for the highlighted best scenario per box, with cost as the tie-breaker: "min_cost" (default; cheapest, ties broken by scenario_name), "min_overflow" (smallest overflow volume) or "max_evapotranspiration" (highest evapotranspiration). In the ">x" box the fewest-overflow scenario is picked first, best_by then breaking ties. "Cost" always refers to the active y_var, so with y_var = "cost_per_evap_pct" the "min_cost" objective picks the scenario with the lowest cost per percentage point of evapotranspiration.
y_var	Character. Which cost measure the y-axis (boxes, points, best markers, frontier) shows: "cost_total" (default; total construction cost, EUR) or "cost_per_evap_pct" (total cost divided by the element evapotranspiration share above the reference minimum , EUR per percentage point – the marginal cost efficiency of evapotranspiration; the baseline comes "for free"). The reference is the lowest evapotranspiration among the scenarios that satisfy the validity criterion (n_overflows <= x; fallback: the complete run when none does) and is named – share, criterion and scenario id – on a second title line. Scenarios

	at or below the reference (including the reference scenario itself) have no defined marginal cost and are dropped from the "cost_per_evap_pct" variant; label_best = TRUE additionally annotates the evapotranspiration gain ("(+NN %Evapotranspiration)") after the price. Titles and the y-axis label switch accordingly.
facet_storage_type	Logical. If TRUE, the plot is split by storage_type into two stacked panels (infiltration box on top, gravel trench below, via <code>ggplot2::facet_grid()</code>), each with its own boxes, best-per-box markers and frontier line; the overlaid points then stay plain circles (the strips already name the type). <code>plotly::ggplotly()</code> keeps the panel split as stacked subplots. Default FALSE.
label_best	Logical. If TRUE, the best scenario per box is annotated next to it: overflow volume plus overflow share ("NN m3 / NN %") for min_overflow, the evapotranspiration share ("NN %") for max_evapotranspiration, or the active y_var value for min_cost – the total cost ("NN EUR") by default, the cost per percentage point of evapotranspiration ("NN EUR/%") with y_var = "cost_per_evap_pct". Default FALSE.
title, lab_x, lab_y	Optional character overrides for the default language-specific title / axis labels.
caption	Character or NULL. Caption below the plot naming the unit-cost rates the EUR values were computed with. NULL (default) uses <code>cost_rates_caption()</code> with the <code>default_cost_rates()</code> ; pass your own string if the costs were computed with different rates, or "" to drop the caption. Note that <code>plotly::ggplotly()</code> drops ggplot captions – re-add it to the interactive version via <code>plotly_add_caption()</code> .
lab_size	Optional character override for the size-legend title.
mark_best	Logical. If TRUE (default), the best scenario per box (see best_by) is highlighted with a black-outlined diamond filled in that box's group colour, so its plotly tooltip inherits the group colour.
connect_best	Logical. If TRUE (default), the highlighted best scenarios of all boxes (overflow counts 0..x plus the ">x" catch-all) are connected by a line – the best-per-overflow-level frontier.
legend_position	Character. Legend position, default "right".

Details

Overflow counts greater than x are collapsed into a single ">x" catch-all box (furthest right, coloured red), keeping the axis readable for the long-tailed 15-year runs (Wien / Bad Aussee reach several hundred overflow events). Its highlighted scenario is the one with the **fewest** overflow events above x (closest to the valid region). Set x high to resolve more counts individually, or to max(n_overflows) to give every count its own box.

The point tooltip is **identical** to `plot_cost_vs_overflow_volume()`: scenario, overflow count / sum (mm) / volume (m3), the element water balance (evapotranspiration / infiltration / overflow, %), the cost breakdown (EUR) and the varying param_grid parameters translated via param_labels. Points and boxes are coloured with the same green (low counts) to red (">x") palette as the sibling plots; because the colour merely echoes the x-axis it carries no separate legend. When both storage types share one panel, the overlaid points are additionally **shaped by the storage type** (filled square

= infiltration box / Sickerbox, filled triangle = gravel trench / Schotterrigol), matching the scatter siblings, and a storage-type legend is shown next to the point-size legend. With `facet_storage_type = TRUE` the plot splits into two stacked storage-type panels instead; the facet strips then carry that information and the points stay **plain circles** for readability. `y_var = "cost_per_evap_pct"` switches the y-axis to the cost per percentage point of evapotranspiration (EUR/%).

The point-size scale is calibrated to the valid region ($0..x$): the extreme overflow volumes of the " $>x$ " catch-all are capped and a minimum size keeps even zero-volume points (the 0 -overflow box) visible, so the many-overflow outliers no longer shrink every valid-region point to an invisible dot. When the storage-type shapes are in use (no faceting), its legend keys are drawn with the storage-type marker (grey; the single present shape, or the square when both types are shown) instead of the default circle; the faceted variant uses circular points and matching circular keys.

Value

A ggplot object. Convert to interactive via `plotly::ggplotly(p, tooltip = "text")`.

See Also

[plot_cost_vs_overflow_volume\(\)](#)

plot_cost_vs_evaporation

Cost vs. evapotranspiration scatter with storage-type shapes

Description

Second companion to [plot_cost_vs_overflow_volume](#) for cost-aware optimisation. Plots the per-scenario **total construction cost** (EUR) on the x-axis against the element **evapotranspiration share** (% of the total water input, from `element.WB_Evapotranspiration_`) on the y-axis. Points are coloured discretely by the **number** of overflow events (same $0..x / >x$ palette used by the sibling plots, legend at the top) and **shaped by the storage type**: filled square = infiltration box (Sickerbox), filled triangle = gravel trench (Schotterrigol).

Usage

```
plot_cost_vs_evaporation(
  simulation_results_optimisation,
  param_grid,
  x = 1,
  filter_n_gtx = FALSE,
  use_jitter = TRUE,
  jitter_width = 0.15,
  jitter_height = 0.15,
  jitter_seed = 1L,
  digits = 2L,
  digits_params = 4L,
  lang = c("de", "en"),
```

```

    param_labels = NULL,
    title = NULL,
    lab_x = NULL,
    lab_y = NULL,
    caption = NULL,
    legend_position = "top"
  )

```

Arguments

simulation_results_optimisation	Data frame with the columns scenario_name, n_overflows, sum_overflows, mulde_area, element.WB_Evapotranspiration_, element.WB_InfiltrationNetto_, element.WB_Oberflaechenablauf_Ueberlauf_, cost_excavation, cost_profiling, cost_filter, cost_storage, cost_total, storage_type. Typically the joined output of add_overflow_events_and_waterbalance() and compute_costs() .
param_grid	Data frame with parameter grid. Must contain scenario_name.
x	Numeric threshold for the overflow-count colour bucket. Values greater than x are pushed into the red ">x" category.
filter_n_gtx	Logical. If TRUE, scenarios with n_overflows > x are dropped before plotting.
use_jitter, jitter_width, jitter_height, jitter_seed	As in plot_wb_tradeoff_overflows() .
digits	Integer. Rounding for numeric values in the tooltip.
digits_params	Integer. Rounding for parameter values in the tooltip.
lang	Character. Plot language: "de" or "en".
param_labels	Named character vector translating param_grid columns to tooltip labels, or NULL to use default_param_labels() for lang.
title, lab_x, lab_y	Optional character overrides for the default language-specific title / axis labels.
caption	Character or NULL. Caption below the plot naming the unit-cost rates the EUR values were computed with. NULL (default) uses cost_rates_caption() with the default_cost_rates() ; pass your own string if the costs were computed with different rates, or "" to drop the caption. Note that <code>plotly::ggplotly()</code> drops ggplot captions – re-add it to the interactive version via plotly_add_caption() .
legend_position	Character. Legend position, default "top".

Details

The tooltip is identical to [plot_cost_vs_overflow_volume\(\)](#): scenario, overflow count / sum (mm) / volume (m3), the element water balance (element.WB_Evapotranspiration_, element.WB_InfiltrationNetto_, element.WB_Oberflaechenablauf_Ueberlauf_, all as % of the total water input), the storage type, the usable storage volume of the storage layer (m3), the cost breakdown (cost_excavation, cost_profiling, cost_filter, cost_storage, cost_total), the derived **cost per percentage point of evapotranspiration** (EUR/%) plus the varying parameters from param_grid (excluding scenario_name).

The plot language can be switched via `lang = "de"` or `lang = "en"`. Titles / axis labels / legend / tooltip labels follow the choice unless explicit overrides are supplied.

Value

A ggplot object. Convert to interactive via `plotly::ggplotly(p, tooltip = "text")`.

See Also

`plot_cost_vs_overflow_volume()` for cost vs. overflow volume and `plot_cost_overflow_boxplot()` for the boxplot views (including `y_var = "cost_per_evap_pct"`, the cost per percentage point of evapotranspiration).

plot_cost_vs_overflow_volume

Cost vs. overflow-volume scatter with n_overflows-coloured points

Description

Companion to `plot_wb_tradeoff_overflows` for cost-aware optimisation. Plots the per-scenario **total construction cost** (EUR) on the x-axis against the **overflow volume** (m3) on the y-axis, with the points coloured discretely by the **number** of overflow events (same 0..x / >x palette used by `plot_wb_tradeoff_overflows`, legend at the top) and **shaped by the storage type**: filled square = infiltration box (Sickerbox), filled triangle = gravel trench (Schotterrigol).

Usage

```
plot_cost_vs_overflow_volume(
  simulation_results_optimisation,
  param_grid,
  x = 1,
  filter_n_gtx = FALSE,
  use_jitter = TRUE,
  jitter_width = 0.15,
  jitter_height = 0.15,
  jitter_seed = 1L,
  digits = 2L,
  digits_params = 4L,
  lang = c("de", "en"),
  param_labels = NULL,
  title = NULL,
  lab_x = NULL,
  lab_y = NULL,
  caption = NULL,
  legend_position = "top"
)
```

Arguments

simulation_results_optimisation	Data frame with the columns scenario_name, n_overflows, sum_overflows, mulde_area, element.WB_Evapotranspiration_, element.WB_InfiltrationNetto_, element.WB_Oberflaechenablauf_Ueberlauf_, cost_excavation, cost_profiling, cost_filter, cost_storage, cost_total, storage_type. Typically the joined output of add_overflow_events_and_waterbalance() and compute_costs() .
param_grid	Data frame with parameter grid. Must contain scenario_name.
x	Numeric threshold for the overflow-count colour bucket. Values greater than x are pushed into the red ">x" category.
filter_n_gtx	Logical. If TRUE, scenarios with n_overflows > x are dropped before plotting.
use_jitter, jitter_width, jitter_height, jitter_seed	As in plot_wb_tradeoff_overflows() .
digits	Integer. Rounding for numeric values in the tooltip.
digits_params	Integer. Rounding for parameter values in the tooltip.
lang	Character. Plot language: "de" or "en".
param_labels	Named character vector translating param_grid columns to tooltip labels, or NULL to use default_param_labels() for lang.
title, lab_x, lab_y	Optional character overrides for the default language-specific title / axis labels.
caption	Character or NULL. Caption below the plot naming the unit-cost rates the EUR values were computed with. NULL (default) uses cost_rates_caption() with the default_cost_rates() ; pass your own string if the costs were computed with different rates, or "" to drop the caption. Note that <code>plotly::ggplotly()</code> drops ggplot captions – re-add it to the interactive version via plotly_add_caption() .
legend_position	Character. Legend position, default "top".

Details

Overflow volume is computed from sum_overflows (in mm on the swale surface, as returned by [add_overflow_events_and_waterbalance\(\)](#)) multiplied by mulde_area (m2) and converted to m3: $\text{overflow_volume_m3} = \text{sum_overflows} * \text{mulde_area} / 1000$.

The tooltip carries the element water balance (element.WB_Evapotranspiration_, element.WB_InfiltrationNetto_, element.WB_Oberflaechenablauf_Ueberlauf_, all as % of the total water input) and the cost breakdown (cost_excavation, cost_profiling, cost_filter, cost_storage, cost_total), the derived **cost per percentage point of evapotranspiration** (EUR/%), the **usable storage volume** of the storage layer (m3; area x height x usable porosity $\theta_{aS} - \theta_{aFC}$, from a storage_volume_m3 column or derived from the storage_theta* columns) plus the varying parameters from param_grid (excluding scenario_name), so the user can hover over a scatter point and see exactly why it landed where it did.

The plot language can be switched via lang = "de" or lang = "en". Titles / axis labels / legend / tooltip labels follow the choice unless explicit overrides are supplied.

Value

A ggplot object. Convert to interactive via `plotly::ggplotly(p, tooltip = "text")`.

See Also

`plot_cost_overflow_boxplot()` for the same data / tooltip shown as a cost-by-overflow-count boxplot and `plot_cost_vs_evaporation()` for cost vs. the element evapotranspiration share.

plot_hpond_vs_ref	<i>Plot the influence of single-parameter variations on a response</i>
-------------------	--

Description

Uses `find_single_param_variations()` (by default excluding `^h_` columns from parameter detection) and plots absolute or percentage deviations of the chosen response. The title shows the reference value in mm.

Usage

```
plot_hpond_vs_ref(  
  data,  
  response,  
  ref_scenario = "s00001",  
  diff = c("abs", "pct"),  
  param_cols = NULL,  
  exclude_cols = NULL,  
  exclude_cols_regex = "^h_",  
  tol = 1e-09,  
  quiet = FALSE  
)
```

Arguments

data	Data frame with at least <code>scenario_name</code> , parameter columns, and response.
response	Character; the response column to plot (e.g. <code>"h_pond_max"</code> or <code>"h_pond_mean"</code>).
ref_scenario	Reference scenario ID (default <code>"s00001"</code>).
diff	<code>"abs"</code> for absolute or <code>"pct"</code> for percentage deviation.
param_cols	Optional explicit parameter columns (recommended if you prefilter).
exclude_cols	Extra columns to exclude from parameter detection.
exclude_cols_regex	Regex to auto-exclude non-parameter columns (default <code>"^h_"</code>).
tol	Numeric tolerance passed through to <code>find_single_param_variations()</code> .
quiet	Logical; diagnostics if FALSE.

Value

A ggplot object.

plot_main_effects	<i>Plot main effects of multiple parameters on an outcome (violin/box/jitter)</i>
-------------------	---

Description

Creates a faceted overview plot showing the distribution of an outcome (for example `n_overflows`) across the tested levels of multiple varied parameters. Parameters are sorted by a simple effect-size proxy: the range of median outcome values across parameter levels.

Usage

```
plot_main_effects(
  df,
  y = "n_overflows",
  params,
  max_levels = 25,
  ylim_lower = 0,
  lang = c("de", "en")
)
```

Arguments

<code>df</code>	A <code>data.frame</code> or <code>tibble</code> containing the outcome column <code>y</code> and the parameter columns listed in <code>params</code> .
<code>y</code>	Character scalar. Name of the outcome column to plot on the y-axis. Defaults to <code>"n_overflows"</code> .
<code>params</code>	Character vector of parameter column names in <code>df</code> to consider.
<code>max_levels</code>	Integer. Parameters with more than <code>max_levels</code> distinct values are dropped to keep the plot readable. Defaults to 25.
<code>ylim_lower</code>	Numeric scalar or <code>NULL</code> . Optional lower display limit for the y-axis. Uses <code>ggplot2::coord_cartesian()</code> so data are not removed before computing violin and boxplots. Defaults to 0.
<code>lang</code>	Character. Plot language: <code>"de"</code> or <code>"en"</code> .

Details

The function is intended for optimisation or sensitivity grids with many parameters, where a single 2D scatter plot is not informative.

Both numeric and character parameters are supported; a character parameter such as `storage_type` gets its own facet panel (its levels are shown as `Sickerbox / Schotterrigol` for `lang = "de"`, `Infiltration box / Gravel trench` for `lang = "en"`).

The plot language can be switched via `lang = "de"` or `lang = "en"`. This affects the title, y-axis label, and selected parameter labels.

Value

A ggplot object with facettted violin, boxplot, and jitter layers.

plot_valid_design_space

Valid solutions in design space ($x \times y$) with overflow-threshold discrete color scale

Description

Visualises model results in a 2D design space (by default mulde_area vs. mulde_height) and highlights valid solutions (by default $n_{\text{overflows}} \leq \text{valid_max}$). The plot is prepared for direct use with `plotly::ggplotly(..., tooltip = "text")` by mapping an HTML tooltip via `aes(text = ...)` that lists all other varied parameters from `param_grid`, excluding `x` and `y`.

Usage

```
plot_valid_design_space(
  param_grid,
  sim_results,
  id_col = "scenario_name",
  overflow_col = "n_overflows",
  valid_max = 1,
  x = "mulde_area",
  y = "mulde_height",
  max_levels = 50,
  alpha_invalid = 0.12,
  size = 2.4,
  digits = 4,
  drop_overflow_gt_valid_max = FALSE,
  jitter = FALSE,
  jitter_width = NULL,
  jitter_height = NULL,
  jitter_factor = 0.005,
  alpha_mode = c("none", "duplicates"),
  alpha_min = 0.2,
  alpha_max = 1,
  keep_param_grid_limits = TRUE,
  facet_storage_type = FALSE,
  lang = c("de", "en"),
  title = NULL,
  subtitle = NULL,
  legend_position = "top"
)
```

Arguments

param_grid	A data.frame/tibble containing the scenario id (id_col) and parameters.
sim_results	A data.frame/tibble containing the scenario id (id_col) and overflow_col.
id_col	Character. Join key (scenario id), default "scenario_name".
overflow_col	Character. Overflow outcome column, default "n_overflows".
valid_max	Numeric. Validity threshold: overflow_col <= valid_max. Also used as the orange breakpoint in the color scale.
x	Character. Name of the x-axis column, default "mulde_area".
y	Character. Name of the y-axis column, default "mulde_height".
max_levels	Integer. Parameter columns with more than max_levels distinct values are ignored in the tooltip list. Default 50.
alpha_invalid	Numeric. Extra alpha multiplier for invalid solutions (background layer).
size	Numeric. Base point size.
digits	Integer. Significant digits for numeric tooltip values.
drop_overflow_gt_valid_max	Logical. If TRUE, scenarios with overflow_col > valid_max are not plotted at all. Default FALSE.
jitter	Logical. If TRUE, apply jitter, useful when many points share identical x/y.
jitter_width	Numeric or NULL. Jitter width in data units. If NULL, defaults from x-range.
jitter_height	Numeric or NULL. Jitter height in data units. If NULL, defaults from y-range.
jitter_factor	Numeric. Factor used to derive default jitter width/height from x/y ranges. Default 0.005.
alpha_mode	Character. Either "none" for fixed alpha or "duplicates" to vary alpha by the number of scenarios sharing the same x/y. Default "none".
alpha_min	Numeric. Minimum alpha used when alpha_mode = "duplicates". Default 0.20.
alpha_max	Numeric. Maximum alpha used when alpha_mode = "duplicates". Default 1.00.
keep_param_grid_limits	Logical. If TRUE and drop_overflow_gt_valid_max = TRUE, the x/y axis limits are fixed to the full range or full set of levels found in param_grid, so the design-space axes do not shrink after filtering. Default TRUE.
facet_storage_type	Logical. If TRUE, the design space is split by storage_type into two stacked panels (infiltration box on top, gravel trench below) with free y-scales, so disjoint per-type levels (e.g. storage_height: 300-1200 mm boxes vs. 900-3600 mm trenches) fill their own panel; duplicate counting for alpha_mode = "duplicates" then happens per panel and the points stay plain circles (the strips already name the type). Requires a storage_type column in param_grid. Without faceting, points are shaped by the storage type (filled square = infiltration box, filled triangle = gravel trench) whenever storage_type varies. Default FALSE.

lang	Character. Plot language: "de" or "en".
title	Character or NULL. Plot title. If NULL, a language-specific default title is used.
subtitle	Character or NULL. Plot subtitle. If NULL, a language-specific default subtitle is used.
legend_position	Character. Legend position, e.g. "top", "bottom", "left", or "right". Default "top".

Details

Varied parameters are detected automatically as columns in `param_grid` with more than one distinct value. Optionally, scenarios with `overflow_col > valid_max` can be removed entirely (`drop_overflow_gt_valid_max = TRUE`). To help identify multiple scenarios that share identical x/y coordinates but differ in other parameters, optional jitter and/or variable transparency can be applied.

Discrete color mapping (threshold-style):

- **dark green** at `n_overflows = 0`
- a discrete palette from dark green to yellow-green for integer levels `1..(valid_max-1)`
- **orange** at `n_overflows = valid_max`
- **red** for `n_overflows > valid_max`, shown as level `">valid_max"`

The plot language can be switched via `lang = "de"` or `lang = "en"`. This affects title, subtitle, legend title, tooltip labels, and selected axis labels.

Value

A ggplot object. Tooltip text is mapped via `aes(text = ...)`.

plot_wb_tradeoff_overflows

Trade-off plot: Infiltration vs. Evapotranspiration (discrete colors by overflow threshold)

Description

Creates a scatter plot from optimisation results with element infiltration on the x-axis and element evapotranspiration on the y-axis.

Usage

```
plot_wb_tradeoff_overflows(
  simulation_results_optimisation,
  param_grid,
  x = 1,
  filter_n_gtx = FALSE,
  use_jitter = TRUE,
```

```

    jitter_width = 0.15,
    jitter_height = 0.15,
    jitter_seed = 1L,
    digits = 2L,
    digits_params = 4L,
    lang = c("de", "en"),
    title = NULL,
    lab_x = NULL,
    lab_y = NULL,
    legend_position = "top"
  )

```

Arguments

simulation_results_optimisation	Data frame with simulation results. Required columns are scenario_name, n_overflows, sum_overflows, element.WB_InfiltrationNetto_, element.WB_Evapotranspiration_ and element.WB_Oberflaechenablauf_Ueberlauf_.
param_grid	Data frame with parameter grid. Must contain scenario_name.
x	Numeric, typically integer. Threshold for overflow coloring. Values greater than or equal to x are mapped to the red category ">=x".
filter_n_gtx	Logical. If TRUE, scenarios with n_overflows >= x are removed before plotting.
use_jitter	Logical. If TRUE, slight jitter is applied to reduce overplotting.
jitter_width, jitter_height	Numeric. Jitter strength in x- and y- direction, only used if use_jitter = TRUE.
jitter_seed	Integer. Seed for reproducible jitter.
digits	Integer. Number of digits used for rounding water balance values in the tooltip.
digits_params	Integer. Number of digits used for rounding numeric parameter values in the tooltip.
lang	Character. Plot language: "de" or "en".
title	Character or NULL. Plot title. If NULL, a language-specific default title is used.
lab_x	Character or NULL. X-axis label. If NULL, a language-specific default label is used.
lab_y	Character or NULL. Y-axis label. If NULL, a language-specific default label is used.
legend_position	Character. Legend position, e.g. "top", "bottom", "left", or "right". Default "top".

Details

Coloring is *discrete* based on n_overflows and a threshold x:

- **dark green** for n_overflows = 0
- a discrete palette from dark green to yellow-green for integer levels 1 . . (x-1)

- **red** for `n_overflows >= x`, shown as category `">=x"`

The plot language can be switched via `lang = "de"` or `lang = "en"`. This affects title, axis labels, legend title, and tooltip labels unless custom labels are supplied explicitly.

Tooltip text additionally includes all parameters from `param_grid` that vary across scenarios, excluding `scenario_name` (translated via `default_param_labels`; mixed numeric / character parameters such as `storage_type` are supported).

If `simulation_results_optimisation` carries a `storage_type` column, the points are additionally **shaped by the storage type** (filled square = infiltration box / Sickerbox, filled triangle = gravel trench / Schotterrigol, as in the cost plots) and the tooltip names the storage type; older single-type result sets plot exactly as before.

Value

A ggplot object.

<code>plotly_add_caption</code>	<i>Add a caption annotation to a ggplotly object</i>
---------------------------------	--

Description

`plotly::ggplotly()` drops ggplot captions (and subtitles). This helper re-adds the caption as a small grey annotation below the plot area (bottom left, under the x-axis title) and widens the bottom margin accordingly. `\n` line breaks are converted to `<br />`.

Usage

```
plotly_add_caption(pl, caption, font_size = 10)
```

Arguments

<code>pl</code>	A plotly object as returned by <code>plotly::ggplotly()</code> .
<code>caption</code>	Character. The caption text; NULL or "" returns <code>pl</code> unchanged.
<code>font_size</code>	Numeric. Caption font size in px. Default 10.

Details

Used by the vignettes together with `cost_rates_caption()` so the interactive cost plots name the unit-cost rates they were computed with.

Value

The modified plotly object.

Examples

```
## Not run:
pl <- plotly::ggplotly(p, tooltip = "text")
pl <- plotly_add_caption(pl, cost_rates_caption("de"))

## End(Not run)
```

plotly_split_legend	<i>Split the combined (colour, shape) ggplotly legend into two clean legends</i>
---------------------	--

Description

plotly::ggplotly() flattens a ggplot with both a colour and a shape aesthetic into one trace per (colour, shape) combination and names the legend entries as tuples such as "(0, Sickerbox / Infiltration box)" – with two storage types and the 0..x / ">x" overflow palette that yields an unreadable legend. This helper post-processes the plotly object:

Usage

```
plotly_split_legend(
  pl,
  lang = c("de", "en"),
  colour_title = NULL,
  shape_title = NULL,
  add_shape_legend = TRUE
)
```

Arguments

pl	A plotly object as returned by plotly::ggplotly(p, tooltip = "text").
lang	Character. "de" or "en"; sets the default legend group titles.
colour_title	Character or NULL. Title of the colour legend group. Defaults to the language-specific "Number of overflow events".
shape_title	Character or NULL. Title of the storage-type legend group. Defaults to the language-specific "Storage type".
add_shape_legend	Logical. If TRUE (default), append the two legend-only storage-type entries.

Details

- the real traces lose their legend entries; instead every overflow class gets one legend-only key drawn as a **neutral circle in the class colour** (a coloured square or triangle would wrongly suggest one specific storage type). The key shares its legend group with the real traces of that class, so clicking it toggles **both** storage types of the class together;

- two legend-only keys (**neutral grey** filled square = infiltration box, filled triangle = gravel trench) are appended under their own **storage-type group title**, so the shape encoding is explained separately from the colours – set `add_shape_legend = FALSE` to skip them (e.g. for storage-type-faceted plots whose strips already label the panels). The keys are **clickable**: since a plotly trace can only carry one legend group (taken by the overflow class), a small JavaScript handler (via `htmlwidgets::onRender()`) toggles all traces drawn with that marker symbol, so each storage type can be shown or hidden individually; the key greys out to reflect the state;
- the combined "colour, shape" legend-title annotation that ggplotly draws over the plot title is removed; group titles take its place and the legend moves to a vertical layout on the right, where the groups stack cleanly.

Traces whose name is not a "(colour, shape)" tuple (frontier lines, best markers, single-aesthetic plots) are left untouched, so the helper is safe to apply to any of the package's interactive plots.

Value

The modified plotly object.

Examples

```
## Not run:
p <- plot_cost_vs_evaporation(sim_results, param_grid, x = 5, lang = "de")
pl <- plotly::ggplotly(p, tooltip = "text")
pl <- plotly_split_legend(pl, lang = "de")

## End(Not run)
```

`read_hdf5_connections` *Read surface-water connections from an HDF5 results file*

Description

Reads an HDF5 file that contains surface-water *connections* (formerly "Verschaltungen"), e.g. a file like `s78853_Verschaltungen.h5`. The file is expected to have:

- a root-level scalar dataset "Anzahl Verschaltungen", and
- one group per connection ("0", "1", ...) that contains
 - scalar metadata (e.g. `von_Layer`, `von_Massnahmenelement`, `nach_Layer`, `nach_Massnahmenelement`) and
 - time-series datasets stored as 2×N matrices (e.g. `Qtatsaechlich`, `delta_h_pond_*`, `delta_theta_*`).

Scalar metadata are returned

- als breite Tabelle (meta),
- als lange Tabelle (scalars) und
- pro Verbindung in einer Unterliste (`connections$group_0, ...`).

Time series werden in Long-Format zurückgegeben und mit den Metadaten verknüpft.

Usage

```
read_hdf5_connections(file)
```

Arguments

file An `hdf5r::H5File` object pointing to a `*_Verschaltungen.h5` file, already opened in read mode.

Value

A named list with components:

n_connections integer(1). Value of the root dataset "Anzahl Verschaltungen" (or NA_integer_ if not present).

meta A tibble with one row per connection: connection_id, from_layer, from_element, to_layer, to_element.

timeseries A tibble with long-format time series for all connections: connection_id, variable, time, value, from_layer, from_element, to_layer, to_element.

scalars A tibble with all scalar datasets per connection in long format: connection_id, variable, value (list), type.

connections A named list of per-connection sublists: connections\$group_0, connections\$group_1, ..., each containing connection_id, meta, timeseries, scalars für genau diese Verbindung.

Examples

```
## Not run:
library(hdf5r)

f <- H5File$new(paths$path_verschaltungen_hdf5, mode = "r")
conn <- read_hdf5_connections(f)

conn$n_connections
conn$meta
conn$timeseries

# Unterliste für die erste Verbindung:
conn$connections$group_0$meta
conn$connections$group_0$timeseries

## End(Not run)
```

read_hdf5_scalars	<i>Read scalar datasets from an HDF5 group</i>
-------------------	--

Description

Reads all scalar datasets contained in an HDF5 group and returns them as a tibble. By default, only numeric / integer scalars are returned. If non-numeric scalars (e.g. strings) are present, they can either be dropped with a warning or kept in a list-column.

Usage

```
read_hdf5_scalars(group, numeric_only = TRUE)
```

Arguments

group	An hdf5r::H5Group object. Direct children of this group are expected to be scalar datasets (i.e. <code>dataset.dims == 0</code>).
numeric_only	Logical (default: TRUE). If TRUE, only numeric / integer scalars are returned (others are dropped with a warning). If FALSE, all scalars are returned in a list-column value together with a type column.

Value

If `numeric_only = TRUE`: A tibble with columns

variable character. Dataset name within the group.

value numeric. Scalar value read from the dataset.

If `numeric_only = FALSE`: A tibble with columns

variable character. Dataset name within the group.

value list. Scalar values (numeric, integer, character, ...).

type character. First class of each value (e.g. "numeric", "character").

Examples

```
## Not run:
# Wasserbilanz: nur numerische Skalare
wb_tbl <- read_hdf5_scalars(res_hdf5_element[["Wasserbilanz"]])

# Metainfo: gemischte Typen (Integer + String)
meta_tbl <- read_hdf5_scalars(res_hdf5_element[["Metainfo"]],
                             numeric_only = FALSE)

## End(Not run)
```

read_hdf5_timeseries *Read HDF5 time series datasets from a group (supports deeperLayers)*

Description

Reads all datasets in an HDF5 group and returns a long tibble with columns: variable, time, value.

Usage

```
read_hdf5_timeseries(  
  ts_groupvariable,  
  deeper_layers_pattern = "deeperLayers|deeper_layers"  
)
```

Arguments

ts_groupvariable
 hdf5r::H5Group
deeper_layers_pattern
 regex to detect deeper-layers datasets

Details

Supported dataset layouts:

- k x N (rows): [1,] = time, [2..k,] = values (series)
- N x k (cols): [, 1] = time, [, 2..k] = values (series)

Special handling for names containing "deeperLayers"/"deeper_layers":

- The value series represent layers below layer 1.
- Output variable names get suffixed with the layer-id: _2, _3, ...
- If the dataset contains only time (no value series), it returns 0 rows.

Value

tibble::tibble(variable, time, value)

read_raindrop_errors *Read RAINDROP error logs into a nested tibble*

Description

Reads the RAINDROP error protocol HDF5 (typically Fehlerprotokol1.h5) for a set of simulation runs and returns a tibble with one row per run and a nested tibble column containing all errors found in the file.

Usage

```
read_raindrop_errors(simulation_names, path_list, debug = FALSE)
```

Arguments

simulation_names	Character vector of simulation run names (e.g. "s00001").
path_list	A path list object passed to <code>kwb.utils::resolve()</code> . Must resolve to an element <code>path_errors_hdf5</code> .
debug	Logical. If TRUE, pass through to <code>kwb.utils::catAndRun()</code> .

Details

The error HDF5 usually contains:

- `/AnzahlFehler` (scalar integer)
- groups `/0`, `/1`, ... each containing datasets such as `Fehlerbeschreibung`, `Layer1`, `Layer2`, `Massnahmenelement1`, `Massnahmenelement2`

Note: In **hdf5r**, HDF5 group names must be addressed as character strings. So group `0` must be accessed as `h5[["0"]]` (not `h5[[0]]`).

Value

A tibble with columns:

- `id` Integer simulation id (parsed from `simulation_names`).
- `scenario_name` The simulation name (e.g. "s00001").
- `path` File path to the error HDF5.
- `file_exists` Logical.
- `number_of_errors` Scalar integer or NA.
- `errors` Nested tibble (list-column) with per-error rows: `error_index`, `Fehlerbeschreibung`, `Layer1`, `Layer2`, `Massnahmenelement1`, `Massnahmenelement2`

Examples

```
## Not run:
errors_df <- read_raindrop_errors(simulation_names, path_list)

# flatten all errors:
all_errors <- tidyr::unnest(errors_df, errors)

## End(Not run)
```

read_site_timeseries	<i>Read and prepare site rain/ET0 time series for the engine (mm/h)</i>
----------------------	---

Description

Factors the time-series preparation duplicated in the Wien and Bad Aussee workflow vignettes into one helper: reads the shipped GeoSphere rain series (rain.csv.gz: columns time (date-time), rr (mm per interval), station, further columns tolerated) and reference ET0 series (et.csv: date;value with dd.mm.yyyy, mm per day), converts both to hours since series start, aligns the series ends (the shorter series is extended to the longer one's end, repeating its last value) and converts the values to the engine's **mm/h** rate convention (rain: mm per interval / interval hours; ET0: mm per day / 24).

Usage

```
read_site_timeseries(path_rain, path_et, verbose = TRUE)
```

Arguments

path_rain	Path to the rain CSV (may be gzipped).
path_et	Path to the ET0 CSV (semicolon separated).
verbose	Print alignment messages (default TRUE).

Value

List with data.frames rain and et (columns time = hours since start, value = mm/h) ready for make_swale_runner(timeseries_rain = , timeseries_et =).

See Also

[make_swale_runner\(\)](#)

run_model

*Run an rainwater management model executable with an input file***Description**

Builds and runs a system command of the form "<path_exe> <path_input>", after normalising both paths to absolute paths. Progress and (optionally) command output are wrapped with `kwb.utils::catAndRun()` for neat logging.

Usage

```
run_model(path_exe, path_input, print_output = FALSE, debug = TRUE, ...)
```

Arguments

path_exe	character(1) Path to the model executable (e.g., a .exe on Windows). The file must exist.
path_input	character(1) Path to the model input file passed as the single argument to the executable. The file must exist.
print_output	logical(1) If TRUE, stream the process output to the console and return the exit status (integer). If FALSE (default), capture and return the command output as a character vector.
debug	logical(1) Forwarded to <code>kwb.utils::catAndRun(dbg = ...)</code> to enable/disable the progress message. Default: TRUE.
...	Additional arguments passed to <code>base::shell()</code> , e.g. <code>timeout</code> on Windows. See <code>?base::shell</code> for details.

Details

Both `path_exe` and `path_input` are converted to absolute, normalised paths via `fs::path_abs()` and `base::normalizePath()`. The command is executed with `base::shell()`, which on Windows invokes the system shell. On non-Windows platforms, prefer `base::system()` if you need full POSIX semantics.

Value

If `print_output = FALSE`, a character vector containing the captured standard output of the command. If `print_output = TRUE`, the (invisible) integer exit status returned by `shell()` (0 indicates success).

Side effects

Executes an external program that may read/write files depending on the model. Ensure you trust the executable and paths provided.

See Also

`base::shell()`, `fs::path_abs()`, `kwb.utils::catAndRun()`

Examples

```
## Not run:
# Example: run a hypothetical model with an input file
exe  <- "C:/path/to/model.exe"
input <- "C:/path/to/input.h5"

# Capture output as character vector
out <- run_model(exe, input, print_output = FALSE)

# Stream output and get exit status
status <- run_model(exe, input, print_output = TRUE)

## End(Not run)
```

run_scenarios	<i>Run scenarios (parallel or sequential) with a user-supplied worker function</i>
---------------	--

Description

Executes scenarios by applying `run_one_scenario` to each element of `indices`. Supports parallel execution via **future.apply** and sequential execution for debugging. Optionally shows progress via **progressr**.

Usage

```
run_scenarios(
  indices,
  run_one_scenario,
  timestep_hours,
  debug = FALSE,
  ...,
  parallel = TRUE,
  workers = parallel::detectCores(),
  show_progress = TRUE,
  progress_handler = "txtprogressbar"
)
```

Arguments

`indices` Vector. Scenario identifiers to iterate over (often integer row indices).

<code>run_one_scenario</code>	Function. Worker function with signature <code>function(x, timestep_hours, debug, ...)</code> . Must accept the arguments <code>timestep_hours</code> and <code>debug</code> .
<code>timestep_hours</code>	Numeric. Time step (hours) forwarded to <code>run_one_scenario</code> .
<code>debug</code>	Logical. Debug flag forwarded to <code>run_one_scenario</code> .
<code>...</code>	Additional arguments forwarded to <code>run_one_scenario</code> .
<code>parallel</code>	Logical. If TRUE, use <code>future.apply::future_lapply</code> . If FALSE, use base <code>lapply</code> .
<code>workers</code>	Integer. Number of workers when <code>parallel = TRUE</code> . Defaults to <code>parallel::detectCores()</code> .
<code>show_progress</code>	Logical. If TRUE, show progress.
<code>progress_handler</code>	Character. Progress handler key. One of "txtprogressbar", "rstudio", "cli".

Value

A list with one element per indices entry containing the return values of `run_one_scenario`.

`sickerbox_level_presets`

Sickerbox storage-height presets (brute force default + manufacturers)

Description

Named list of storage-height level vectors (mm) for the infiltration-box storage layer. `brute_force` is the default used by the workflow vignettes (300/600/900/1200 mm – itself a combination of several box types). The manufacturer presets are generated with `stack_levels()` from typical module heights of commercial block systems; verify against the current data sheets before productive optimisation runs.

Usage

```
sickerbox_level_presets(max_height = 2600)
```

Arguments

`max_height` Maximum total stack height in mm passed to `stack_levels()` (default 2600).

Value

Named list of sorted numeric vectors (mm).

stack_levels	<i>Achievable storage-layer stack heights from module heights</i>
--------------	---

Description

Enumerates all storage-layer heights that can be built by stacking (and mixing) the given module heights, e.g. full blocks combined with at most one half block.

Usage

```
stack_levels(modules, max_count = rep(7L, length(modules)), max_height = 2600)
```

Arguments

modules	Numeric vector of module heights in mm (e.g. c(660, 350) for a full block plus a half block).
max_count	Integer vector (recycled to length(modules)): maximum number of modules of each type in one stack. Defaults to 7 for every module (cf. GRAF EcoBloc smart, stackable up to 7 layers).
max_height	Maximum total stack height in mm (default 2600).

Value

Sorted numeric vector of achievable stack heights in mm.

Examples

```
stack_levels(360) # 360, 720, ..., 2520
stack_levels(c(660, 350), max_count = c(7, 1)) # Rigofill full + half block
```

Index

`add_overflow_events_and_waterbalance`,
 [3](#), [15](#)
`add_overflow_events_and_waterbalance()`,
 [7](#), [30](#), [33](#), [35](#)

`base::normalizePath()`, [50](#)
`base::shell()`, [50](#), [51](#)
`base::system()`, [50](#)

`compute_costs`, [4](#)
`compute_costs()`, [7](#), [25](#), [28](#), [30](#), [33](#), [35](#)
`cost_rates_caption`, [6](#)
`cost_rates_caption()`, [31](#), [33](#), [35](#), [42](#)

`default_canonical_wb_variables`, [7](#)
`default_cost_rates`, [7](#)
`default_cost_rates()`, [5](#), [6](#), [25](#), [27](#), [31](#), [33](#),
 [35](#)
`default_param_labels`, [8](#), [42](#)
`default_param_labels()`, [30](#), [33](#), [35](#)
`default_storage_spec`, [9](#)
`default_storage_spec()`, [24–26](#), [28](#)
`default_storage_types`, [9](#)
`default_storage_types()`, [9](#), [22](#)
`download_engine`, [10](#)

`find_min_feasible`, [11](#)
`find_min_feasible()`, [23–25](#)
`find_single_param_variations`, [13](#)
`fs::path_abs()`, [50](#), [51](#)

`get_simulation_results_all`, [14](#)
`get_simulation_results_optim`, [15](#), [17](#)
`get_simulation_results_optim_parallel`,
 [16](#)

`h5_ensure_dataset`, [17](#)
`h5_ensure_datasets_from_values`, [18](#)
`h5_read_values`, [19](#)
`h5_validate_write`, [19](#)
`h5_write_values`, [20](#)

`hdf5r::H5File`, [45](#)
`hdf5r::H5Group`, [46](#)

`kwb.utils::catAndRun()`, [50](#), [51](#)

`list_h5_datasets`, [21](#)

`make_swale_runner`, [22](#)
`make_swale_runner()`, [24–26](#), [28](#), [49](#)

`optimise_swale_design`, [23](#)
`optimise_swale_design()`, [23](#), [25](#), [28](#)
`optimise_swale_design_simultaneous`, [25](#)
`optimise_swale_design_simultaneous()`,
 [25](#)

`plot_cost_overflow_boxplot`, [29](#)
`plot_cost_overflow_boxplot()`, [6](#), [8](#), [34](#),
 [36](#)
`plot_cost_vs_evaporation`, [32](#)
`plot_cost_vs_evaporation()`, [6](#), [36](#)
`plot_cost_vs_overflow_volume`, [32](#), [34](#)
`plot_cost_vs_overflow_volume()`, [6](#), [8](#), [29](#),
 [31–34](#)
`plot_hpond_vs_ref`, [36](#)
`plot_main_effects`, [37](#)
`plot_valid_design_space`, [38](#)
`plot_wb_tradeoff_overflows`, [34](#), [40](#)
`plot_wb_tradeoff_overflows()`, [7](#), [33](#), [35](#)
`plotly_add_caption`, [42](#)
`plotly_add_caption()`, [6](#), [31](#), [33](#), [35](#)
`plotly_split_legend`, [43](#)

`read_hdf5_connections`, [44](#)
`read_hdf5_scalars`, [15](#), [16](#), [46](#)
`read_hdf5_timeseries`, [15](#), [16](#), [47](#)
`read_raindrop_errors`, [48](#)
`read_site_timeseries`, [49](#)
`resolve`, [15](#), [16](#)
`run_model`, [50](#)
`run_scenarios`, [51](#)

sickerbox_level_presets, [52](#)

stack_levels, [53](#)

stack_levels(), [52](#)

tools::R_user_dir, [10](#)