Package: ssdsims (via r-universe)

June 4, 2026

Title What the Package Does (One Line, Title Case)

Version 0.0.0.9011

Description What the package does (one paragraph).

License Apache License (== 2.0) | file LICENSE

URL https://poissonconsulting.github.io/ssdsims/

BugReports https://github.com/poissonconsulting/ssdsims/issues

Depends R (>= 4.1)

Imports chk, dplyr, dqrng (>= 0.4.1), methods, parallel, purrr, rlang, ssdtools (>= 2.6.0), tibble, tidyr, withr

Suggests knitr, quarto, readr, ssddata, testthat (>= 3.0.0)

VignetteBuilder quarto

Remotes poissonconsulting/ssdtools@dev

Config/testthat/edition 3

Encoding UTF-8

Roxygen list(markdown = TRUE)

Config/roxygen2/version 8.0.0.9000

Config/pak/sysreqs libicu-dev libjpeg-dev libpng-dev libxml2-dev libssl-dev libx11-dev

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

Date/Publication 2026-06-04 16:23:42 UTC

RemoteUrl https://github.com/poissonconsulting/ssdsims

RemoteRef HEAD

RemoteSha df24d3a3d7196d09bc75c1ab10aafc298b488e24

Contents

`local_dqrng_backend` *Local dqrng pcg64 Backend*

Description

Activates the dqrng `pcg64` RNG backend for the duration of the calling frame, then resets it when `.local_envir` exits. While active, base R's `runif()`, `rnorm()`, `rbinom()`, `rexp()`, `rgamma()`, `rpois()`, `sample.int()`, and `sample()` (and therefore `dplyr::slice_sample()` and `ssdtools::ssd_r*()`) draw from dqrng's `pcg64`, seeded via `dqrng::dqset.seed()`. `pcg64` is forced explicitly because it accepts the length-2 `stream` argument the per-task primer design relies on; dqrng's own default (`Xoroshiro128++`) does not.

Usage

```
local_dqrng_backend(.local_envir = parent.frame())
```

Arguments

`.local_envir`	`[environment]` The environment to use for scoping.

Details

Registering the backend is a process-global side effect that also advances base R's `.Random.seed`. `local_dqrng_backend()` follows the withr convention (compare `withr::local_seed()`): it pairs activation with deferred reset so the backend is always restored, including on error.

The helper is reentrant. `dqrng::register_methods()` / `dqrng::restore_methods()` keep a single global save-slot, so a nested reset would tear the backend down for the still-open outer scope. To avoid this, a `local_dqrng_backend()` call made while the backend is already active is a no-op: it does not re-activate the backend and schedules no further reset. Only the outermost call activates the backend on entry and resets it on exit, so the RNG stream is identical whether or not a nested call occurs.

Value

Invisibly returns TRUE if this call activated the backend (the outermost scope) or FALSE if the backend was already active and the call was a no-op.

See Also

withr::local_seed(), dqrng::dqset.seed().

Examples

```
local_dqrng_backend()
dqrng::dqset.seed(42, stream = c(1L, 2L))
runif(3)
```

---

local_dqrng_state *Local/With dqrng State*

Description

local_dqrng_state() installs a per-task (seed, primer) starting point as the running dqrng RNG state via dqrng::dqset.seed(), restoring the previous state when .local_envir exits. with_dqrng_state() evaluates code with that state installed, then restores the previous state. The primer argument is the per-task primer (the value handed to dqrng's stream argument, per TARGETS-DESIGN.md §2 and the GLOSSARY); the _state suffix marks that the wrapper installs that primer as the running RNG state.

Usage

```
local_dqrng_state(seed, primer, .local_envir = parent.frame())

with_dqrng_state(seed, primer, code)
```

Arguments

seed	[whole number] A scalar seed passed to dqrng::dqset.seed().
primer	[integer(2)] A length-2 integer primer passed as the stream argument of dqrng::dqset.seed(). NA_integer_ is permitted (the reserved INT_MIN encoding of TARGETS-DESIGN.md §2).
.local_envir	[environment] The environment to use for scoping.
code	[any] Code to execute in the temporary environment

Details

These are the dqrng-path analogues of `local_lecuyer_cmrg_state()` / `with_lecuyer_cmrg_state()`. Like those helpers they snapshot the RNG state on entry (via `dqrng::dqrng_get_state()`) and `withr::defer()` a restore (via `dqrng::dqrng_set_state()`), so a call leaves the surrounding RNG stream undisturbed, including on error.

Both require an active dqrng backend: they abort unless a `local_dqrng_backend()` scope is open. This fails fast rather than silently seeding base R's Mersenne-Twister.

Value

`local_dqrng_state()` invisibly returns `primer`; `with_dqrng_state()` returns the value of `code`.

See Also

`withr::local_seed()`, `local_dqrng_backend()`, `local_lecuyer_cmrg_state()`.

Examples

```
local_dqrng_backend()
local_dqrng_state(42, c(1L, 2L))
runif(3)

with_dqrng_state(42, c(1L, 2L), runif(3))
```

`local_lecuyer_cmrg_seed`

Local/With L'Ecuyer-CMRG Seed

Description

`local_lecuyer_cmrg_seed()` seeds the L'Ecuyer-CMRG RNG with a scalar integer via `base::set.seed()`, restoring the previous state when `.local_envir` exits. `with_lecuyer_cmrg_seed()` evaluates `code` with that seed in effect, then restores the previous state. For a `.Random.seed`-style state vector (e.g. from `get_lecuyer_cmrg_stream_state()` or `parallel::nextRNGStream()`) use `local_lecuyer_cmrg_state()` / `with_lecuyer_cmrg_state()`.

Usage

```
local_lecuyer_cmrg_seed(seed, .local_envir = parent.frame())

with_lecuyer_cmrg_seed(seed, code)
```

Arguments

seed	[integer(1)] The random seed to use to evaluate the code.
.local_envir	[environment] The environment to use for scoping.
code	[any] Code to execute in the temporary environment

Value

with_lecuyer_cmrg_seed() returns the value of code.

See Also

withr::local_seed(), local_lecuyer_cmrg_state(), parallel::nextRNGStream().

Examples

```
local_lecuyer_cmrg_seed(42)
runif(3)

with_lecuyer_cmrg_seed(42, {
  runif(3)
})
```

---

local_lecuyer_cmrg_state

Local/With L'Ecuyer-CMRG State

Description

local_lecuyer_cmrg_state() sets the L'Ecuyer-CMRG RNG state to a .Random.seed-style integer vector (length 7) by assigning to .Random.seed directly, restoring the previous state when .local_envir exits. with_lecuyer_cmrg_state() evaluates code with that state in effect, then restores the previous state. A *state* is the full internal RNG state (as returned by parallel::nextRNGStream() or get_lecuyer_cmrg_stream_state()); contrast with base::set.seed() which takes a scalar *seed* (see local_lecuyer_cmrg_seed() / with_lecuyer_cmrg_seed()).

Usage

```
local_lecuyer_cmrg_state(state, .local_envir = parent.frame())

with_lecuyer_cmrg_state(state, code)
```

Arguments

state	[integer(7)] A L'Ecuyer-CMRG .Random.seed vector.
.local_envir	[environment] The environment to use for scoping.
code	[any] Code to execute in the temporary environment

Value

local_lecuyer_cmrg_state() invisibly returns state; with_lecuyer_cmrg_state() returns the value of code.

See Also

parallel::nextRNGStream(), local_lecuyer_cmrg_seed().

Examples

```
state <- with_lecuyer_cmrg_seed(42, parallel::nextRNGStream(.Random.seed))
local_lecuyer_cmrg_state(state)
runif(3)

with_lecuyer_cmrg_state(state, runif(3))
```

---

ssd_data *Assemble and Validate Datasets for a Simulation Scenario*

Description

Collects one or more datasets into a validated, named collection - the single entry point through which ssd_define_scenario() takes dataset input. Each dataset must carry a numeric Conc column (the species sensitivity distribution convention); additional columns are preserved.

Usage

```
ssd_data(...)
```

Arguments

...	One or more data frames, optionally named. Each is validated for a numeric Conc column.

Details

Names are taken from the argument names where supplied, otherwise derived from the argument expression by symbol capture (e.g. `ssddata::ccme_boron` becomes `"ccme_boron"`). A literal with no derivable name (e.g. a bare `data.frame(...)` call) must be given an explicit name.

`ssd_data()` is intended to grow: the planned `scenario-input-types` change (see `TARGETS-DESIGN.md` section 12) will let each input also be one of the data *generators* `ssd_run_scenario()` accepts today - a `fitdists` or `tmbfit` object, a generator function, or a function-name string - with the data materialised by the dataset registry. For now each input must be a data frame.

Value

An `ssdsims_data` object: a named list of validated tibbles.

Examples

```
ssd_data(ssddata::ccme_boron)
ssd_data(boron = ssddata::ccme_boron, cadmium = ssddata::ccme_cadmium)
```

---

ssd_define_scenario *Define a Simulation Scenario*

Description

Constructs a purely declarative `ssdsims_scenario` object: the root of the targets-based pipeline (see `TARGETS-DESIGN.md` section 1). The object stores only declarative fields - a scalar `seed`, the replicate count `nsim`, the sample sizes `nrow`, the dataset *names*, and the `fit` and `hc` argument grids. It performs **no** random-number generation, **no** task expansion, and has **no** dependency on `targets`.

Usage

```
ssd_define_scenario(
  data,
  nsim,
  seed,
  ...,
  name = NULL,
  nrow = 6L,
  replace = FALSE,
  dists = ssdtools::ssd_dists_bcanz(),
  rescale = FALSE,
  computable = FALSE,
  at_boundary_ok = TRUE,
  min_pmix = list(ssdtools::ssd_min_pmix),
  range_shape1 = list(c(0.05, 20)),
  range_shape2 = list(c(0.05, 20)),
  proportion = 0.05,
  ci = FALSE,
```

```
  nboot = 1000,
  est_method = "multi",
  ci_method = "weighted_samples",
  parametric = TRUE,
  partition_by = NULL,
  upload = NULL
)
```

Arguments

data	An ssd_data() collection (preferred), or - for convenience - a single data frame or a (named or unnamed) list of data frames. Bare inputs are validated via the same Conc contract as ssd_data().
nsim	A count of the number of data sets to generate.
seed	A scalar whole number; the scenario's RNG root. Required - changing it fully re-roots the scenario's random-number draws.
...	Unused; must be empty.
name	An optional dataset name for the single-data-frame form, overriding the derived name. Must not be combined with a named list or an ssd_data() collection.
nrow	A positive whole number of the minimum number of non-missing rows.
replace	A logical vector specifying whether to sample with replacement.
dists	A character vector of the distribution names.
rescale	A flag specifying whether to leave the values unchanged (FALSE) or to rescale concentration values by dividing by the geometric mean of the minimum and maximum positive finite values (TRUE) or a string specifying whether to leave the values unchanged ("no") or to rescale concentration values by dividing by the geometric mean of the minimum and maximum positive finite values ("geomean") or to logistically transform ("odds").
computable	A flag specifying whether to only return fits with numerically computable standard errors.
at_boundary_ok	A flag specifying whether a model with one or more parameters at the boundary should be considered to have converged (default = TRUE).
min_pmix	The min_pmix function(s), referenced **by name**. Supply either a character vector of names, or a function (or list of functions) with a single argument that inputs the number of rows of data and returns a proportion between 0 and 0.5 - in which case the name is derived from the argument expression (e.g. ssdtools::ssd_min_pmix gives "ssd_min_pmix"), mirroring dataset name derivation. Only the name is stored; the function is resolved later via the min_pmix registry (a future roadmap step).
range_shape1	A list of numeric vectors of length two of the lower and upper bounds for the shape1 parameter.
range_shape2	A list of numeric vectors of length two of the lower and upper bounds for the shape2 parameter.
proportion	A numeric vector of proportion values to estimate hazard concentrations for.

ci	A flag specifying whether to estimate confidence intervals (by bootstrapping).
nboot	A count of the number of bootstrap samples to use to estimate the confidence limits. A value of 10,000 is recommended for official guidelines.
est_method	A string specifying whether to estimate directly from the model-averaged cumulative distribution function (est_method = 'multi') or to take the arithmetic mean of the estimates from the individual cumulative distribution functions weighted by the AICc derived weights (est_method = 'arithmetic') or or to use the geometric mean instead (est_method = 'geometric').
ci_method	A string specifying which method to use for estimating the standard error and confidence limits from the bootstrap samples. The default and recommended value is still ci_method = "weighted_samples" which takes bootstrap samples from each distribution proportional to its AICc based weights and calculates the confidence limits (and SE) from this single set. ci_method = "multi_fixed" and ci_method = "multi_free" generate the bootstrap samples using the model-averaged cumulative distribution function but differ in whether the model weights are fixed at the values for the original dataset or re-estimated for each bootstrap sample dataset. The value ci_method = "MACL" (was ci_method = "weighted_arithmetic"), which is only included for historical reasons, takes the weighted arithmetic mean of the confidence limits while ci_method = GMACL which takes the weighted geometric mean of the confidence limits was added for completeness but is also not recommended. Finally ci_method = "arithmetic_samples" and ci_method = "geometric_samples" take the weighted arithmetic or geometric mean of the values for each bootstrap iteration across all the distributions and then calculate the confidence limits (and SE) from the single set of samples.
parametric	A flag specifying whether to perform parametric bootstrapping as opposed to non-parametrically resampling the original data with replacement.
partition_by	An optional named list with data, fit, and hc character vectors naming the Hive partition axes per step. When NULL the documented per-step defaults are used.
upload	An optional upload specification (a list), or NULL for no upload.

Details

Input data is forwarded through ssd_data() for validation (a numeric Conc column is required) and retained on the scenario (as $data) so a local run (ssd_run_scenario_baseline()) can sample it directly. The dataset *names* ($datasets) are what the targets/cluster path hashes and resolves through the registry, so that path need not carry the data frames.

Value

An S3 object of class ssdsims_scenario.

Dataset input

The preferred form is an ssd_data() collection, which owns validation and naming: ssd_define_scenario(ssd_data(bor = ccme_boron, cadmium = ccme_cadmium), ...). For convenience, bare data frame input is also accepted in four forms (routed through the same Conc validation):

1. A single data frame, name derived from the argument expression: `ssd_define_scenario(ssddata::ccme_boron, ...)` gives `"ccme_boron"`.
2. A single data frame with an explicit `name=`: `ssd_define_scenario(ssddata::ccme_boron, name = "boron", ...)`.
3. A named list, names taken from the list: `ssd_define_scenario(list(boron = ccme_boron, cadmium = ccme_cadmium), ...)`.
4. An unnamed list, names derived per element: `ssd_define_scenario(list(ccme_boron, ccme_cadmium), ...)`.

Supplying both a named list and `name=` is an error.

`ci = FALSE`

When `ci = FALSE` is the only confidence-interval value, the bootstrap-only knobs `nboot`, `ci_method`, and `parametric` are meaningless. Passing any of them in that case is an error; set `ci = c(FALSE, TRUE)` to enable bootstrap, or omit the knobs.

Examples

```
ssd_define_scenario(ssddata::ccme_boron, nsim = 100L, nrow = c(5L, 10L), seed = 42L)
```

`ssd_fit_dists_sims` *Fit SSD Distributions to Simulated Data*

Description

Fit SSD Distributions to Simulated Data

Usage

```
ssd_fit_dists_sims(
  x,
  dists = ssdtools::ssd_dists_bcanz(),
  ...,
  rescale = FALSE,
  computable = FALSE,
  at_boundary_ok = TRUE,
  min_pmix = list(ssdtools::ssd_min_pmix),
  range_shape1 = list(c(0.05, 20)),
  range_shape2 = range_shape1,
  seed = NULL,
  silent = TRUE,
  .progress = FALSE
)
```

Arguments

x	A data frame with sim and stream integer columns and a list column of the data frames to fit distributions to.
dists	A character vector of the distribution names.
...	Additional arguments passed to ssdtools::ssd_fit_dists().
rescale	A flag specifying whether to leave the values unchanged (FALSE) or to rescale concentration values by dividing by the geometric mean of the minimum and maximum positive finite values (TRUE) or a string specifying whether to leave the values unchanged ("no") or to rescale concentration values by dividing by the geometric mean of the minimum and maximum positive finite values ("geomean") or to logistically transform ("odds").
computable	A flag specifying whether to only return fits with numerically computable standard errors.
at_boundary_ok	A flag specifying whether a model with one or more parameters at the boundary should be considered to have converged (default = TRUE).
min_pmix	A list of one or more functions with a single argument that inputs the number of rows of data and returns a proportion between 0 and 0.5.
range_shape1	A list of numeric vectors of length two of the lower and upper bounds for the shape1 parameter.
range_shape2	A list of numeric vectors of length two of the lower and upper bounds for the shape2 parameter.
seed	An integer of the starting seed or NULL.
silent	A flag indicating whether fits should fail silently.
.progress	Whether to show a purrr::progress bar.

Value

The x tibble with a list column fits of fistdist objects.

---

ssd_hc_sims *Estimate hazard concentrations for multiple simulations using bootstrapping*

Description

Estimate hazard concentrations for multiple simulations using bootstrapping

Usage

```
ssd_hc_sims(
  x,
  proportion = 0.05,
  ...,
  ci = FALSE,
  nboot = 1000,
  est_method = "multi",
  ci_method = "weighted_samples",
  parametric = TRUE,
  seed = NULL,
  save_to = NULL,
  .progress = FALSE
)
```

Arguments

x	A data frame with sim and stream integer columns and a list column of fitdists objects.
proportion	A numeric vector of proportion values to estimate hazard concentrations for.
...	Additional arguments passed to `ssdtools::ssd_hc()`.
ci	A flag specifying whether to estimate confidence intervals (by bootstrapping).
nboot	A count of the number of bootstrap samples to use to estimate the confidence limits. A value of 10,000 is recommended for official guidelines.
est_method	A string specifying whether to estimate directly from the model-averaged cumulative distribution function (`est_method = 'multi'`) or to take the arithmetic mean of the estimates from the individual cumulative distribution functions weighted by the AICc derived weights (`est_method = 'arithmetic'`) or or to use the geometric mean instead (`est_method = 'geometric'`).
ci_method	A string specifying which method to use for estimating the standard error and confidence limits from the bootstrap samples. The default and recommended value is still `ci_method = "weighted_samples"` which takes bootstrap samples from each distribution proportional to its AICc based weights and calculates the confidence limits (and SE) from this single set. `ci_method = "multi_fixed"` and `ci_method = "multi_free"` generate the bootstrap samples using the model-averaged cumulative distribution function but differ in whether the model weights are fixed at the values for the original dataset or re-estimated for each bootstrap sample dataset. The value `ci_method = "MACL"` (was `ci_method = "weighted_arithmetic"`), which is only included for historical reasons, takes the weighted arithmetic mean of the confidence limits while `ci_method = GMACL` which takes the weighted geometric mean of the confidence limits was added for completeness but is also not recommended. Finally `ci_method = "arithmetic_samples"` and `ci_method = "geometric_samples"` take the weighted arithmetic or geometric mean of the values for each bootstrap iteration across all the distributions and then calculate the confidence limits (and SE) from the single set of samples.
parametric	A flag specifying whether to perform parametric bootstrapping as opposed to non-parametrically resampling the original data with replacement.

seed	An integer of the starting seed or NULL.
save_to	NULL or a string specifying a directory to save where the bootstrap datasets and parameter estimates (when successfully converged) to.
.progress	Whether to show a purrr::progress bar.

Value

The x tibble with a list column hc of data frames produced by applying ssd_hc() to fits.

`ssd_run_scenario` *Run Scenario*

Description

Run Scenario

Usage

```
ssd_run_scenario(x, ...)

## S3 method for class 'data.frame'
ssd_run_scenario(
  x,
  ...,
  nrow = 6L,
  replace = FALSE,
  dists = ssdtools::ssd_dists_bcanz(),
  rescale = FALSE,
  computable = FALSE,
  at_boundary_ok = TRUE,
  min_pmix = list(ssdtools::ssd_min_pmix),
  range_shape1 = list(c(0.05, 20)),
  range_shape2 = list(c(0.05, 20)),
  proportion = 0.05,
  ci = FALSE,
  nboot = 1000,
  est_method = "multi",
  ci_method = "weighted_samples",
  parametric = TRUE,
  seed = NULL,
  nsim = 100L,
  stream = getOption("ssdsims.stream", 1L),
  start_sim = 1L,
  .progress = FALSE
)

## S3 method for class 'fitdists'
```

```
ssd_run_scenario(
  x,
  ...,
  nrow = 6L,
  dist_sim = "top",
  dists = ssdtools::ssd_dists_bcanz(),
  rescale = FALSE,
  computable = FALSE,
  at_boundary_ok = TRUE,
  min_pmix = list(ssdtools::ssd_min_pmix),
  range_shape1 = list(c(0.05, 20)),
  range_shape2 = list(c(0.05, 20)),
  proportion = 0.05,
  ci = FALSE,
  nboot = 1000,
  est_method = "multi",
  ci_method = "weighted_samples",
  parametric = TRUE,
  seed = NULL,
  nsim = 100L,
  stream = getOption("ssdsims.stream", 1L),
  start_sim = 1L,
  .progress = FALSE
)

## S3 method for class 'tmbfit'
ssd_run_scenario(
  x,
  ...,
  nrow = 6L,
  dists = ssdtools::ssd_dists_bcanz(),
  rescale = FALSE,
  computable = FALSE,
  at_boundary_ok = TRUE,
  min_pmix = list(ssdtools::ssd_min_pmix),
  range_shape1 = list(c(0.05, 20)),
  range_shape2 = list(c(0.05, 20)),
  proportion = 0.05,
  ci = FALSE,
  nboot = 1000,
  est_method = "multi",
  ci_method = "weighted_samples",
  parametric = TRUE,
  seed = NULL,
  nsim = 100L,
  stream = getOption("ssdsims.stream", 1L),
  start_sim = 1L,
  .progress = FALSE
```

```
)

## S3 method for class 'character'
ssd_run_scenario(
  x,
  ...,
  nrow = 6L,
  args = list(),
  dists = ssdtools::ssd_dists_bcanz(),
  rescale = FALSE,
  computable = FALSE,
  at_boundary_ok = TRUE,
  min_pmix = list(ssdtools::ssd_min_pmix),
  range_shape1 = list(c(0.05, 20)),
  range_shape2 = list(c(0.05, 20)),
  proportion = 0.05,
  ci = FALSE,
  nboot = 1000,
  est_method = "multi",
  ci_method = "weighted_samples",
  parametric = TRUE,
  seed = NULL,
  nsim = 100L,
  stream = getOption("ssdsims.stream", 1L),
  start_sim = 1L,
  .progress = FALSE
)

## S3 method for class '`function`'
ssd_run_scenario(
  x,
  ...,
  nrow = 6L,
  args = list(),
  dists = ssdtools::ssd_dists_bcanz(),
  rescale = FALSE,
  computable = FALSE,
  at_boundary_ok = TRUE,
  min_pmix = list(ssdtools::ssd_min_pmix),
  range_shape1 = list(c(0.05, 20)),
  range_shape2 = list(c(0.05, 20)),
  proportion = 0.05,
  ci = FALSE,
  nboot = 1000,
  est_method = "multi",
  ci_method = "weighted_samples",
  parametric = TRUE,
  seed = NULL,
```

```
  nsim = 100L,
  stream = getOption("ssdsims.stream", 1L),
  start_sim = 1L,
  .progress = FALSE
)
```

Arguments

x	The object to use for the scenario.
...	Unused.
nrow	A positive whole number of the minimum number of non-missing rows.
replace	A logical vector specifying whether to sample with replacement.
dists	A character vector of the distribution names.
rescale	A flag specifying whether to leave the values unchanged (FALSE) or to rescale concentration values by dividing by the geometric mean of the minimum and maximum positive finite values (TRUE) or a string specifying whether to leave the values unchanged ("no") or to rescale concentration values by dividing by the geometric mean of the minimum and maximum positive finite values ("geomean") or to logistically transform ("odds").
computable	A flag specifying whether to only return fits with numerically computable standard errors.
at_boundary_ok	A flag specifying whether a model with one or more parameters at the boundary should be considered to have converged (default = TRUE).
min_pmix	A number between 0 and 0.5 specifying the minimum proportion in mixture models.
range_shape1	A numeric vector of length two of the lower and upper bounds for the shape1 parameter.
range_shape2	shape2 parameter.
proportion	A numeric vector of proportion values to estimate hazard concentrations for.
ci	A flag specifying whether to estimate confidence intervals (by bootstrapping).
nboot	A count of the number of bootstrap samples to use to estimate the confidence limits. A value of 10,000 is recommended for official guidelines.
est_method	A string specifying whether to estimate directly from the model-averaged cumulative distribution function (`est_method = 'multi'`) or to take the arithmetic mean of the estimates from the individual cumulative distribution functions weighted by the AICc derived weights (`est_method = 'arithmetic'`) or or to use the geometric mean instead (`est_method = 'geometric'`).
ci_method	A string specifying which method to use for estimating the standard error and confidence limits from the bootstrap samples. The default and recommended value is still `ci_method = "weighted_samples"` which takes bootstrap samples from each distribution proportional to its AICc based weights and calculates the confidence limits (and SE) from this single set. `ci_method = "multi_fixed"` and `ci_method = "multi_free"` generate the bootstrap samples using the model-averaged cumulative distribution function but differ in whether the model weights

	are fixed at the values for the original dataset or re-estimated for each bootstrap sample dataset. The value ci_method = "MACL" (was ci_method = "weighted_arithmetic"), which is only included for historical reasons, takes the weighted arithmetic mean of the confidence limits while ci_method = GMACL which takes the weighted geometric mean of the confidence limits was added for completeness but is also not recommended. Finally ci_method = "arithmetic_samples" and ci_method = "geometric_samples" take the weighted arithmetic or geometric mean of the values for each bootstrap iteration across all the distributions and then calculate the confidence limits (and SE) from the single set of samples.
parametric	A flag specifying whether to perform parametric bootstrapping as opposed to non-parametrically resampling the original data with replacement.
seed	An integer of the starting seed or NULL.
nsim	A count of the number of data sets to generate.
stream	A count of the stream number.
start_sim	A count of the number of the simulation to start from.
.progress	Whether to show a purrr::progress bar.
dist_sim	A character vector specifying the distributions in the fitdists object or "all"`` for all the distributio to treat the distributions as a single distribution.
args	A named list of the argument values.

Value

A tibble of nested data sets.

Methods (by class)

- ssd_run_scenario(data.frame): Run scenario using data.frame to sample data
- ssd_run_scenario(fitdists): Run scenario using fitdists object to generate data
- ssd_run_scenario(tmbfit): Run scenario using tmbfit object to generate data
- ssd_run_scenario(character): Run scenario using name of function to generate sequence of random numbers
- ssd_run_scenario(`function`): Run scenario data using function to generate sequence of random numbers

Examples

```
ssd_run_scenario(ssddata::ccme_boron, nsim = 2)

fit <- ssdtools::ssd_fit_dists(ssddata::ccme_boron)
ssd_run_scenario(fit, dist_sim = c("lnorm", "top"), nsim = 3)

fit <- ssdtools::ssd_fit_dists(ssddata::ccme_boron)
ssd_run_scenario(fit[[1]], nsim = 3)

ssd_run_scenario("rlnorm", nsim = 3)

ssd_run_scenario(ssdtools::ssd_rlnorm, nsim = 3)
```

ssd_run_scenario_baseline *Run a Scenario with the Baseline Loop Runner*

Description

Executes the three task tables in dependency order - `sample`, `fit`, then `hc` - by looping over each table with `purrr::pmap()` and looking up each task's parent result by the parent's `<step>_id` foreign key. The `fit` step truncates its parent sample inline (`head(sample, nrow)`) before fitting. The runner does no task expansion of its own (it consumes `ssd_scenario_tasks()`); it just threads outputs forward and returns the collected per-step results.

Usage

```
ssd_run_scenario_baseline(scenario)
```

Arguments

`scenario`	An `ssdsims_scenario` from `ssd_define_scenario()`.

Details

This is the no-frills baseline: it runs in-process, with **no** `targets` dependency, **no** shard grouping or `partition_by`, and **no** Parquet I/O.

It **is** reproducible without an external seed. The runner opens one `local_dqrng_backend()` scope and seeds each `sample`/`fit`/`hc` task exactly once through its `*_data_task_primer()` wrapper, with `seed = scenario$seed` and a per-task primer derived from the task's canonical identity (`task_primer()` over the `task_axes(step)` columns). Because each task's `(seed, primer)` pair fully determines its RNG starting point, two runs of a scenario with a fixed `seed` yield identical results, and a task's result is independent of the order in which tasks run. These same `*_data_task_primer()` wrappers are the per-task entry point a future `targets` shard body and the replay helper (`TARGETS-DESIGN.md` §7) reuse.

The scenario retains the data frames it was built from, so the runner reads them directly - no separate `data` argument. `min_pmix` names are resolved against `ssdtools` until the registry roadmap step lands.

Value

A named list with `sample`, `fit`, and `hc` elements: each the corresponding task table augmented with a list column of per-task results (`sample` draws, `fits` objects, and `hc` tibbles).

Examples

```
scenario <- ssd_define_scenario(
  ssddata::ccme_boron,
  nsim = 1L,
  nrow = 6L,
```

```
  seed = 42L,
  dists = "lnorm"
)
out <- ssd_run_scenario_baseline(scenario)
out$hc
```

`ssd_scenario_fit_tasks` *Derive the fit Task Table from a Scenario*

Description

Crosses each sample-task identity (`dataset`, `sim`, `replace`) with the scenario's `nrow` values and each row of the scenario's `fit` argument grid (`rescale`, `computable`, `at_boundary_ok`, `min_pmix` name, `range_shape1`, `range_shape2`). `nrow` is a genuine `fit` cross-join axis: the `fit` step truncates its parent sample inline (`head(sample, nrow)`, RNG-free) before fitting, so the shared draw is sub-truncated without a separate `data` step (`TARGETS-DESIGN.md` section 5). Parent-identity columns are preserved verbatim so the table can be grouped directly downstream. `min_pmix` is referenced by name, not by function value (`TARGETS-DESIGN.md` section 1.1).

Usage

```
ssd_scenario_fit_tasks(scenario)
```

Arguments

`scenario`	An `ssdsims_scenario` from `ssd_define_scenario()`.

Details

Each row carries a `fit_id` primary key and a `sample_id` foreign key referencing its parent sample task.

Value

An `ssdsims_tasks` object recording the `"fit"` step, with one row per `(dataset, sim, replace, nrow)` identity crossed with the fit grid.

Examples

```
scenario <- ssd_define_scenario(
  ssddata::ccme_boron,
  nsim = 3L,
  seed = 42L,
  rescale = c(FALSE, TRUE)
)
ssd_scenario_fit_tasks(scenario)
```

`ssd_scenario_hc_tasks` *Derive the hc Task Table from a Scenario*

Description

Crosses each fit-task identity with each row of the scenario's `hc` argument grid (`nboot`, `est_method`, `ci_method`, `parametric`). The expansion honours the construction-time `ci = FALSE` collapse (`TARGETS-DESIGN.md` section 1.2): rows where `ci = FALSE` are not multiplied across the bootstrap-only knobs (`nboot`, `ci_method`, `parametric`), which are stored as `NA`, while `ci = TRUE` rows fan out across the full grid.

Usage

```
ssd_scenario_hc_tasks(scenario)
```

Arguments

`scenario`	An `ssdsims_scenario` from `ssd_define_scenario()`.

Details

Each row carries an `hc_id` primary key and a `fit_id` foreign key referencing its parent fit task.

Value

An `ssdsims_tasks` object recording the `"hc"` step, with one row per fit-task identity crossed with the (collapsed) hc grid.

Examples

```
scenario <- ssd_define_scenario(
  ssddata::ccme_boron,
  nsim = 2L,
  seed = 42L,
  ci = c(FALSE, TRUE),
  nboot = c(10L, 100L)
)
ssd_scenario_hc_tasks(scenario)
```

ssd_scenario_sample_tasks *Derive the sample Task Table from a Scenario*

Description

Expands an ssdsims_scenario into the sample task table: one row per cell of the cross-join of the scenario's dataset names, replicate index (1:nsim), and replace values. Each row is the single random draw of n_max = max(nrow) rows that every nrow value sub-truncates (TARGETS-DESIGN.md section 5), so nrow is **not** a sample axis - the draw is shared. n_max is carried as an ordinary integer column. The derivation performs no random-number generation and adds no seed/primer/stream columns (those arrive in later roadmap steps; see TARGETS-DESIGN.md section 2).

Usage

```
ssd_scenario_sample_tasks(scenario)
```

Arguments

scenario	An ssdsims_scenario from ssd_define_scenario().

Details

Each row carries a path-style sample_id primary key.

Value

An ssdsims_tasks object (a classed tibble recording the "sample" step) with one row per (dataset, sim, replace) cell, a sample_id key, and a carried n_max column.

Examples

```
scenario <- ssd_define_scenario(ssddata::ccme_boron, nsim = 3L, seed = 42L)
ssd_scenario_sample_tasks(scenario)
```

ssd_scenario_tasks *Expand a Scenario into all Three Task Tables*

Description

The canonical expansion entry point (TARGETS-DESIGN.md section 1/section 2): derives the sample, fit, and hc task tables from a scenario in one call and bundles them into an ssdsims_task_set. The per-step derivations (ssd_scenario_sample_tasks(), ssd_scenario_fit_tasks(), ssd_scenario_hc_tasks()) remain available for callers that need a single table.

Usage

```
ssd_scenario_tasks(scenario)
```

Arguments

scenario	An ssdsims_scenario from ssd_define_scenario().

Value

An ssdsims_task_set object: a list with sample, fit, and hc elements, each an ssdsims_tasks table.

Examples

```
scenario <- ssd_define_scenario(ssddata::ccme_boron, nsim = 3L, seed = 42L)
tasks <- ssd_scenario_tasks(scenario)
tasks
tasks$hc
```

---

ssd_sim_data *Generate Data for Simulations*

Description

A family of functions to generate a tibble of nested data sets.

Usage

```
ssd_sim_data(x, ...)

## S3 method for class 'data.frame'
ssd_sim_data(
  x,
  ...,
  nrow = 6L,
  replace = FALSE,
  seed = NULL,
  nsim = 100L,
  stream = getOption("ssdsims.stream", 1L),
  start_sim = 1L,
  .progress = FALSE
)

## S3 method for class 'fitdists'
ssd_sim_data(
  x,
  ...,
```

```
  nrow = 6L,
  dist_sim = "top",
  seed = NULL,
  nsim = 100L,
  stream = getOption("ssdsims.stream", 1L),
  start_sim = 1L,
  .progress = FALSE
)

## S3 method for class 'tmbfit'
ssd_sim_data(
  x,
  ...,
  nrow = 6L,
  seed = NULL,
  nsim = 100L,
  stream = getOption("ssdsims.stream", 1L),
  start_sim = 1L,
  .progress = FALSE
)

## S3 method for class 'character'
ssd_sim_data(
  x,
  ...,
  nrow = 6L,
  args = list(),
  seed = NULL,
  nsim = 100L,
  stream = getOption("ssdsims.stream", 1L),
  start_sim = 1L,
  .progress = FALSE
)

## S3 method for class '`function`'
ssd_sim_data(
  x,
  ...,
  nrow = 6L,
  args = list(),
  seed = NULL,
  nsim = 100L,
  stream = getOption("ssdsims.stream", 1L),
  start_sim = 1L,
  .progress = FALSE
)
```

Arguments

x	The object to use for generating the data.
...	Unused.
nrow	A numeric vector of the number of rows in the generated data which must be between 5 and 1,000,
replace	A logical vector specifying whether to sample with replacement.
seed	An integer of the starting seed or NULL.
nsim	A count of the number of data sets to generate.
stream	A count of the stream number.
start_sim	A count of the number of the simulation to start from.
.progress	Whether to show a purrr::progress bar.
dist_sim	A character vector specifying the distributions in the fitdists object or "all"`` for all the distributio to treat the distributions as a single distribution.
args	A named list of the argument values.

Value

A tibble of nested data sets.

Methods (by class)

- ssd_sim_data(data.frame): Generate data by sampling from data.frame
- ssd_sim_data(fitdists): Generate data from fitdists object
- ssd_sim_data(tmbfit): Generate data from tmbfit object
- ssd_sim_data(character): Generate data using name of function
- ssd_sim_data(`function`): Generate data using function to generate sequence of random numbers

Examples

```
ssd_sim_data(ssddata::ccme_boron, nrow = 5, nsim = 3)

fit <- ssdtools::ssd_fit_dists(ssddata::ccme_boron)
ssd_sim_data(fit, nrow = 5, nsim = 3)

fit <- ssdtools::ssd_fit_dists(ssddata::ccme_boron)
ssd_sim_data(fit[[1]], nrow = 5, nsim = 3)

ssd_sim_data("rnorm", nrow = 5, nsim = 3)

ssd_sim_data(ssdtools::ssd_rlnorm, nrow = 5, nsim = 3)
```

task_primer *Derive a Per-task Primer from its Parameters*

Description

Derives the per-task **primer** – a length-2 integer vector – from `rlang::hash(params)`, suitable for the `stream` argument of `dqrng::dqset.seed()`. Together with the scenario seed, the primer fully specifies a task's RNG starting point: `dqrng::dqset.seed(seed, stream = task_primer(params))`. It pairs with `local_dqrng_state()`, which installs the `(seed, primer)` pair under an active `local_dqrng_backend()` scope.

Usage

```
task_primer(params)
```

Arguments

`params`	A plain named list of task parameters, or a single-row data frame (one task-table row).

Details

The primer packs 64 bits of the `rlang::hash()` digest (xxhash128) as `c(hi32, lo32)`. Each 32-bit half is encoded as a signed int32, with the reserved bit pattern `0x80000000` (INT_MIN, which R cannot represent as a non-`NA` integer) mapped to `NA_integer_`; dqrng accepts `NA_integer_` in `stream` and treats it as INT_MIN, so the encoding recovers the full 64 bits of stream entropy.

`params` may be a plain named list or a single-row data frame (one row of a `{sample,fit,hc}_tasks` table). A data-frame row is normalised to a canonical plain list – the inverse of `tibble::tibble_row()` – by dropping all attributes, unwrapping length-1 list-style columns to their element, and leaving df-style (nested data-frame) columns as data frames, before hashing. The primer is therefore identical whether derived from the row or from the equivalent plain list. Note that `rlang::hash()` is order-sensitive, so the plain list must use the **same name order** as the task-table columns to reproduce the row's primer (assembling `params` in a canonical column order is part of the `task-tables` caller contract below).

`task_primer()` normalises **structure, not meaning**: it hashes whatever `params` it is given. The canonical, name-keyed representation is a caller contract assembled where `params` is built (`task-tables`, over the `task-lists` tables). Per the three-step model the RNG-consuming steps each take a primer over their task identity:

- **sample** – keyed `(dataset, sim, replace)` only. `nrow` is deliberately absent: every `nrow` shares one `n_max`-row draw that the `fit` step truncates inline (`head(sample, nrow)`, RNG-free, no separate primer), so excluding `nrow` is load-bearing for the sub-truncation property (`TARGETS-DESIGN.md` §5).
- **fit** – the parent `sample` identity plus `nrow` and the fit-grid row (`rescale`, `computable`, `at_boundary_ok`, `min_pmix` name, `range_shape1`, `range_shape2`). `nrow` IS part of the fit primer: a fit on a different truncation is a genuinely different computation.

- **hc** – the parent `fit` identity plus the hc-grid row (`ci`, `nboot`, `est_method`, `ci_method`, `parametric`).

Function-valued parameters (e.g. `min_pmix`) MUST be referenced **by name**, not by function value, so a recompile or JIT does not move a task's primer.

Value

An integer vector of length 2 – the per-task primer – to pass as the `stream` argument of `dqrng::dqset.seed()` (via `local_dqrng_state()`).

See Also

`local_dqrng_state()`, `local_dqrng_backend()`.

Examples

```
task_primer(list(dataset = "boron", sim = 1L, replace = FALSE))
```

Index