# Package 'pmsims'

September 17, 2026

**Title** Simulation-Based Sample Size Tools for Prediction Models

**Version** 1.0.0

**Description** Provides a flexible, simulation-based toolkit for exploring how much data are needed to develop reliable prediction models. It works by repeatedly generating data, fitting models, and evaluating performance to show how sample size affects predictive accuracy, calibration, and overfitting. The package supports continuous, binary, and time-to-event outcomes and can be used with both regression-based modelling approaches and machine-learning methods. It is designed to help researchers plan studies, assess feasibility, and build more robust and generalisable models. The methods are described in Olaniran et al. (2026) <doi:10.1186/s12874-026-02935-9> and Shamsutdinova et al. (2026) <doi:10.48550/arXiv.2602.23507>.

**License** GPL (>= 3)

**URL** https://pmsims-package.github.io/pmsims/, https://github.com/pmsims-package/pmsims

**BugReports** https://github.com/pmsims-package/pmsims/issues

**Depends** R (>= 4.1.0)

**Imports** cli, ggplot2, lifecycle, mlpwr, pROC, stats, survival, timeROC, utils

**Suggests** covr, DescTools, doParallel, foreach, glmnet, knitr, mlbench, mlr, randomForestSRC, ranger, rmarkdown, synthpop, testthat (>= 3.0.0), tuneRanger, xgboost

**Encoding** UTF-8

**LazyData** true

**LazyDataCompression** xz

**VignetteBuilder** knitr

**Config/testthat/edition** 3

**Config/roxygen2/version** 8.0.0

**NeedsCompilation** no

**Author** Ewan Carr [aut, cre] (ORCID: <https://orcid.org/0000-0002-1146-4922>),
Gordon Forbes [aut] (ORCID: <https://orcid.org/0000-0001-8662-3636>),
Ridwan Olaniran [aut] (ORCID: <https://orcid.org/0000-0001-7342-8639>),
Diana Shamsutdinova [aut] (ORCID:
<https://orcid.org/0000-0003-2434-3641>),
Daniel Stahl [aut] (ORCID: <https://orcid.org/0000-0001-7987-6619>),
Sarah Markham [aut] (ORCID: <https://orcid.org/0000-0002-8755-5935>),
Felix Zimmer [aut] (ORCID: <https://orcid.org/0000-0002-8127-0007>)

**Maintainer** Ewan Carr <ewan.carr@kcl.ac.uk>

**Repository** CRAN

**Date/Publication** 2026-09-17 09:30:02 UTC

# Contents



---

`precomputed` *Precomputed sample-size simulations*

---

## Description

Results from four completed sample-size searches, one for each of the package's entry points. A search repeatedly simulates datasets, fits a prediction model to each and evaluates its performance across a range of candidate sample sizes, so a realistic run takes minutes to hours. These objects were computed once, with the calls shown under *Source*, so that the examples and the vignette can demonstrate the output without recomputing it.

## Usage

```
binary_example

continuous_example

survival_example

custom_example
```

## Format

A list of class "pmsims". The most useful components are:

`min_n` The estimated minimum sample size.

`perf_n` Expected performance at `min_n`.

`target_performance` The performance target the search aimed for.

`metric` The performance metric used.

`model` The model fitted to each simulated dataset.

`mlpwr_ds` The sampled designs and simulated performance values behind the fitted curve, used by `plot()`.

`summaries` Aggregated performance summaries across replications.

`simulation_time` Elapsed time of the original run, in seconds.

Objects produced by the wrapper functions additionally record the data-generating configuration (for example `outcome_prevalence`, `correlation` and `complexity`); see `simulate_binary()` for the full set.

An object of class pmsims of length 36.

An object of class pmsims of length 35.

An object of class pmsims of length 37.

An object of class pmsims of length 24.

## Details

Each object is a "pmsims" object, as returned by `simulate_binary()`, `simulate_continuous()`, `simulate_survival()` and `simulate_custom()`, and can be used with `print()` and `plot()` in the usual way.

## Source

Generated by `data-raw/precomputed-examples.R`.

`binary_example`:

```
set.seed(123)
simulate_binary(
  signal_parameters = 20,
  noise_parameters = 0,
  complexity = 1,
  data_control = list(correlation = 0.3),
  outcome_prevalence = 0.30,
  maximum_achievable_cstatistic = 0.80,
  model = "glm",
  metric = "calibration_slope",
  target_performance = 0.85,
  n_reps_total = 1000,
  mean_or_assurance = "assurance"
)
```

continuous_example:

```
set.seed(123)
simulate_continuous(
  signal_parameters = 15,
  noise_parameters = 0,
  complexity = 1,
  data_control = list(correlation = 0.3),
  maximum_achievable_rsquared = 0.50,
  model = "lm",
  metric = "calibration_slope",
  target_performance = 0.95,
  n_reps_total = 1000,
  mean_or_assurance = "assurance"
)
```

survival_example:

```
set.seed(123)
simulate_survival(
  signal_parameters = 15,
  noise_parameters = 0,
  complexity = 1,
  data_control = list(correlation = 0.3),
  maximum_achievable_cindex = 0.70,
  baseline_hazard = 0.01,
  censoring_rate = 0.30,
  model = "coxph",
  metric = "calibration_slope",
  target_performance = 0.90,
  n_reps_total = 1000,
  mean_or_assurance = "assurance"
)
```

custom_example: a linear model with three independent predictors and a population $R^2$ of 0.5, targeting a calibration slope of 0.9 with 80% assurance. The search uses bounds of 25 to 1,000 participants, 1,000 replications and 30,000 test observations. See simulate_custom() for the generating functions and executable call (seed 123).

## See Also

simulate_binary(), simulate_continuous(), simulate_survival(), simulate_custom()

## Examples

```
binary_example
binary_example$min_n
```

simulate_binary *Minimum sample size for binary-outcome prediction models*

## Description

Compute the minimum sample size required to develop a prediction model with a binary outcome. The function wraps a simulation-based engine that combines a bisection search with Gaussian-process curve fitting. From user inputs (outcome prevalence, maximum achievable performance, target performance, etc.) it constructs a data-generating function, a model-fitting function, and a metric function, then searches for the smallest $n$ that meets the chosen performance criterion.

## Usage

```
simulate_binary(
  signal_parameters,
  noise_parameters = 0,
  complexity = 1,
  data_control = NULL,
  outcome_prevalence,
  maximum_achievable_cstatistic,
  model = c("glm", "lasso", "ridge", "rf", "xgboost"),
  metric = "calibration_slope",
  target_performance,
  n_reps_total = 1000,
  mean_or_assurance = "assurance",
  ...
)
```

## Arguments

`signal_parameters`
: Integer. Number of candidate predictors associated with the outcome (i.e., true signal features).

`noise_parameters`
: Integer. Number of candidate predictors not associated with the outcome (noise features). Default is 0.

`complexity`
: Integer in 1:4 selecting the data-generating signal structure (see *Data control*). Default 1.

`data_control`
: Optional named list controlling the predictors (see *Data control*). Default `NULL` (generator defaults).

`outcome_prevalence`
: Numeric in (0, 1). Target prevalence of the binary outcome in the intended modelling context.

`maximum_achievable_cstatistic`
: Numeric in (0, 1). Maximum achievable C-statistic with effectively unlimited data. This is used to calibrate the data-generating mechanism and is not the minimum acceptable threshold.

`model` Character string specifying the modelling algorithm. One of `"glm"` (logistic regression), `"lasso"`, `"ridge"`, `"rf"` (random forest), or `"xgboost"` (gradient-boosted trees).

`metric` Character string naming the performance metric used to assess the sample size; defaults to `"calibration_slope"`. Metric identifiers use one canonical form throughout the package, such as `"calibration_slope"`, `"calibration_in_the_large"`, `"auc"`, `"r2"`, `"cindex"`, and `"csse"`.

`"calibration_slope"` is the slope from regressing the observed outcome on the model's linear predictor in held-out data; 1 indicates perfect calibration, and values below 1 indicate overfitting. Note that for the machine-learning models (`"lasso"`, `"ridge"`, `"rf"`, `"xgboost"`) this is converted internally to the calibration slope squared error for optimisation and translated back before results are returned; you don't need to do anything, and `target_performance` is still given on the calibration slope scale. Results derived this way carry a footnote marker in the printed output.

`"csse"` is the calibration slope squared error, $-(1 - s)^2$ for a calibration slope $s$, so that larger is better and 0 is perfect calibration. It can be requested directly, which is mainly useful for advanced use and for comparison against the internal conversion described above. When requesting it directly you are responsible for supplying `target_performance` on the CSSE scale: a calibration slope target of `0.9` corresponds to a CSSE target of `-0.01`. No adjustment is applied on your behalf, and results are reported on the CSSE scale.

`target_performance` Numeric. Minimum acceptable value of the selected performance metric $M^*$; the algorithm searches for the smallest $n$ meeting the chosen criterion with respect to this threshold.

`n_reps_total` Integer. Total number of simulation replications used by the engine across the search.

`mean_or_assurance` Character string, either `"mean"` or `"assurance"`. Controls whether the minimum $n$ is defined by the mean-based criterion or the assurance-based criterion (with the assurance level $\delta$ controlled by the engine's defaults or additional arguments in `...`).

`...` Additional options passed to `simulate_custom()` (e.g., assurance level $\delta$, per-iteration settings).

## Value

An object of class `"pmsims"` containing the estimated minimum sample size and simulation diagnostics (inputs, fitted GP curve, intermediate evaluations, and summary metrics).

## Criteria

Two formulations are supported.

- **Mean-based**: find the smallest $n$ such that the expected model performance exceeds the target $M^*$, i.e.

$$\min_n \ \mathbb{E}_{D_n}\{M \mid D_n\} \geq M^*.$$

- **Assurance-based**: find the smallest $n$ such that the probability the performance exceeds $M^*$ is at least $\delta$ (e.g. 0.80), i.e.

$$\min_n \ \mathbb{P}_{D_n}(M \mid D_n \geq M^*) \geq \delta.$$

Here, $M$ is the chosen performance metric and the probability/expectation is over repeated samples of training data of size $n$. The assurance criterion explicitly accounts for variability across training sets; models with higher variance typically require larger $n$ to satisfy it.

## Data control

`complexity` selects the signal structure of the data-generating mechanism: `1` purely linear, `2` linear + quadratic, `3` linear + quadratic + interaction, `4` the Friedman function. `data_control` is an optional list fine-tuning the predictors:

`nonlinear_strength` Numeric in `[0, 1)`. Fraction of the signal variance carried by the nonlinear, linearly-inaccessible component. Applies to complexity 2 and 3 only; ignored (with a warning) for 1 and 4. If omitted, the generator's per-complexity default is used.

`correlation` Numeric in $[-1, 1]$. Pairwise correlation among the candidate predictors. Default `0.3`.

`predictor_distribution` One of `"normal"`, `"uniform"`, `"binary"`, `"exponential"`, `"lognormal"`, `"t"`, `"laplace"`. `"binary"` selects 0/1 predictors and requires `binary_predictor_prevalence`; any other value selects continuous predictors from that family. Default `"normal"`.

`binary_predictor_prevalence` Numeric in `(0, 1)`. Prevalence of the binary predictors; required when `predictor_distribution = "binary"`, ignored (with a warning) otherwise. Note: binary predictors are incompatible with complexity 2/3 because squaring a 0/1 variable returns itself.

## See Also

`simulate_continuous()`, `simulate_survival()`, `simulate_custom()`

## Examples

```
set.seed(123)
est <- simulate_binary(
  signal_parameters = 3,
  noise_parameters = 0,
  complexity = 1,
  data_control = list(correlation = 0),
  outcome_prevalence = 0.50,
  maximum_achievable_cstatistic = 0.80,
  model = "glm",
  metric = "calibration_slope",
  target_performance = 0.9,
  mean_or_assurance = "assurance",
  min_sample_size = 50,
  max_sample_size = 1000,
  n_reps_total = 1000,
  test_n = 30000,
```

```
  progress = FALSE
)
est
est$min_n
plot(est)
```

---

`simulate_continuous` *Minimum sample size for continuous-outcome prediction models*

---

## Description

Compute the minimum sample size required to develop a prediction model with a **continuous** outcome. This wraps the same simulation engine as `simulate_binary()`, combining bisection search with Gaussian-process learning-curve modelling. From user inputs (maximum achievable performance, target performance, etc.) it constructs a data-generating function, model-fitting function, and metric function, then searches for the smallest $n$ meeting the chosen criterion.

## Usage

```
simulate_continuous(
  signal_parameters,
  noise_parameters = 0,
  complexity = 1,
  data_control = NULL,
  maximum_achievable_rsquared,
  model = c("lm", "lasso", "ridge", "rf", "xgboost"),
  metric = "calibration_slope",
  target_performance,
  n_reps_total = 1000,
  mean_or_assurance = "assurance",
  ...
)
```

## Arguments

`signal_parameters`
: Integer. Number of candidate predictors associated with the outcome (i.e., true signal features).

`noise_parameters`
: Integer. Number of candidate predictors not associated with the outcome (noise features). Default is 0.

`complexity`
: Integer in 1:4 selecting the data-generating signal structure (see *Data control*). Default `1`.

`data_control`
: Optional named list controlling the predictors (see *Data control*). Default `NULL` (generator defaults).

`maximum_achievable_rsquared`
: Numeric in (0, 1). Maximum achievable $R^2$ with effectively unlimited data. This is used to calibrate the data-generating mechanism and is not the minimum acceptable threshold.

`model`
: Character string specifying the modelling algorithm. One of `"lm"` (linear regression), `"lasso"`, `"ridge"`, `"rf"` (random forest), or `"xgboost"` (gradient-boosted trees).

`metric`
: Character string naming the performance metric used to assess the sample size; defaults to `"calibration_slope"`. Metric identifiers use one canonical form throughout the package, such as `"calibration_slope"`, `"calibration_in_the_large"`, `"auc"`, `"r2"`, `"cindex"`, and `"csse"`.

  `"calibration_slope"` is the slope from regressing the observed outcome on the model's linear predictor in held-out data; 1 indicates perfect calibration, and values below 1 indicate overfitting. Note that for the machine-learning models (`"lasso"`, `"ridge"`, `"rf"`, `"xgboost"`) this is converted internally to the calibration slope squared error for optimisation and translated back before results are returned; you don't need to do anything, and `target_performance` is still given on the calibration slope scale. Results derived this way carry a footnote marker in the printed output.

  `"csse"` is the calibration slope squared error, $-(1 - s)^2$ for a calibration slope $s$, so that larger is better and 0 is perfect calibration. It can be requested directly, which is mainly useful for advanced use and for comparison against the internal conversion described above. When requesting it directly you are responsible for supplying `target_performance` on the CSSE scale: a calibration slope target of `0.9` corresponds to a CSSE target of `-0.01`. No adjustment is applied on your behalf, and results are reported on the CSSE scale.

`target_performance`
: Numeric. Minimum acceptable value of the selected performance metric $M^*$; the algorithm searches for the smallest $n$ meeting the chosen criterion with respect to this threshold.

`n_reps_total`
: Integer. Total number of simulation replications used by the engine across the search.

`mean_or_assurance`
: Character string, either `"mean"` or `"assurance"`. Controls whether the minimum $n$ is defined by the mean-based criterion or the assurance-based criterion (with the assurance level $\delta$ controlled by the engine's defaults or additional arguments in `...`).

`...`
: Additional options passed to `simulate_custom()` (e.g., assurance level $\delta$, per-iteration settings).

## Value

An object of class `"pmsims"` containing the estimated minimum sample size and simulation diagnostics (inputs, fitted GP curve, intermediate evaluations, and summary metrics).

## Criteria

Two formulations are supported.

- **Mean-based**: find the smallest $n$ such that the expected model performance exceeds the target $M^*$, i.e.

$$\min_n \ \mathbb{E}_{D_n}\{M \mid D_n\} \geq M^*.$$

- **Assurance-based**: find the smallest $n$ such that the probability the performance exceeds $M^*$ is at least $\delta$ (e.g. 0.80), i.e.

$$\min_n \ \mathbb{P}_{D_n}(M \mid D_n \geq M^*) \geq \delta.$$

Here, $M$ is the chosen performance metric and the probability/expectation is over repeated samples of training data of size $n$. The assurance criterion explicitly accounts for variability across training sets; models with higher variance typically require larger $n$ to satisfy it.

## Data control

`complexity` selects the signal structure of the data-generating mechanism: `1` purely linear, `2` linear + quadratic, `3` linear + quadratic + interaction, `4` the Friedman function. `data_control` is an optional list fine-tuning the predictors:

`nonlinear_strength` Numeric in `[0, 1)`. Fraction of the signal variance carried by the nonlinear, linearly-inaccessible component. Applies to complexity 2 and 3 only; ignored (with a warning) for 1 and 4. If omitted, the generator's per-complexity default is used.

`correlation` Numeric in $[-1, 1]$. Pairwise correlation among the candidate predictors. Default `0.3`.

`predictor_distribution` One of `"normal"`, `"uniform"`, `"binary"`, `"exponential"`, `"lognormal"`, `"t"`, `"laplace"`. `"binary"` selects 0/1 predictors and requires `binary_predictor_prevalence`; any other value selects continuous predictors from that family. Default `"normal"`.

`binary_predictor_prevalence` Numeric in `(0, 1)`. Prevalence of the binary predictors; required when `predictor_distribution = "binary"`, ignored (with a warning) otherwise. Note: binary predictors are incompatible with complexity 2/3 because squaring a 0/1 variable returns itself.

## See Also

`simulate_binary()`, `simulate_survival()`, `simulate_custom()`

## Examples

```
set.seed(123)
est <- simulate_continuous(
  signal_parameters = 3,
  noise_parameters = 0,
  complexity = 1,
  data_control = list(correlation = 0),
  maximum_achievable_rsquared = 0.50,
  model = "lm",
  metric = "calibration_slope",
  target_performance = 0.9,
  mean_or_assurance = "assurance",
  min_sample_size = 50,
```

```
  max_sample_size = 1000,
  n_reps_total = 1000,
  test_n = 30000,
  progress = FALSE
)
est
est$min_n
plot(est)
```

---

`simulate_custom` *Minimum sample size for custom simulation workflows*

---

## Description

Compute the minimum sample size required to achieve a target level of predictive performance using user-defined simulation components. `simulate_custom()` is the low-level interface in `pmsims`: users supply a data-generating function, a model-fitting function, and a metric function, and the chosen search engine estimates the smallest $n$ meeting the selected performance criterion.

## Usage

```
simulate_custom(
  data_function,
  model_function,
  metric_function,
  target_performance,
  c_statistic = NULL,
  mean_or_assurance = "assurance",
  test_n = 30000,
  min_sample_size = NULL,
  max_sample_size = NULL,
  n_reps_total = 1000,
  n_reps_per = 20,
  method = "mlpwr",
  progress = TRUE,
  verbose = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| `data_function` | Function taking a single argument, `n`, giving the training sample size, and returning a dataset that can be passed to `model_function`. |
| `model_function` | Function that fits a model to the dataset returned by `data_function`. It must take the generated dataset as its only argument and return a fitted model object. |

`metric_function`
: Function that evaluates predictive performance on test data. It must take three positional arguments in the order `(test_data, fitted_model, model_name)` and return a single numeric value. Optionally, users may set `attr(metric_function, "value_on_error")` to a single numeric fallback value to be returned if model fitting or metric evaluation fails during a simulation run.

`target_performance`
: Numeric target value for the chosen performance metric. The search aims to find the smallest sample size $n$ for which the selected criterion is met relative to this threshold.

`c_statistic`
: Optional numeric value used only by the internal start-value heuristics for some outcome and metric combinations. In most custom workflows this should be left as `NULL`.

`mean_or_assurance`
: Character string specifying the criterion used to define the minimum sample size. Must be either `"mean"` or `"assurance"`.

`test_n`
: Integer size of the test dataset used to evaluate model performance. This should usually be large enough that test-set variability is negligible relative to the training-sample search.

`min_sample_size`
: Optional integer lower bound for the sample-size search. If supplied, `max_sample_size` must also be supplied.

`max_sample_size`
: Optional integer upper bound for the sample-size search. If supplied, `min_sample_size` must also be supplied. Supplying both bounds defines the search space directly, so the adaptive starting-value search is skipped. Because the runtime estimate is extrapolated from that stage, no long-run warning is issued either.

`n_reps_total`
: Integer total number of simulation replications allocated to the search. The search evaluates approximately `n_reps_total / n_reps_per` candidate sample sizes.

`n_reps_per`
: Integer number of simulation replications performed at each candidate sample size.

`method`
: Character string specifying the search engine. Defaults to `"mlpwr"`.

`progress`
: Logical flag controlling whether the `mlpwr` progress bar is shown for `mlpwr`-based methods.

`verbose`
: Logical flag controlling engine-specific diagnostic output when supported. For the bisection engine, setting `verbose = TRUE` stores the iteration history on the returned object.

`...`
: Additional arguments passed to the selected search engine.

## Value

An object of class `"pmsims"` containing the estimated minimum sample size.

## See Also

`simulate_binary()`, `simulate_continuous()`, `simulate_survival()`

## Examples

```
# Three independent predictors with a population R-squared of 0.5.
data_fun <- function(n) {
  x1 <- rnorm(n)
  x2 <- rnorm(n)
  x3 <- rnorm(n)
  y <- (x1 + x2 + x3) / sqrt(3) + rnorm(n)
  data.frame(y = y, x1 = x1, x2 = x2, x3 = x3)
}

model_fun <- function(dat) {
  stats::lm(y ~ ., data = dat)
}

# Calibration slope evaluated on independent test data.
metric_fun <- function(test_data, fit, model) {
  preds <- stats::predict(fit, newdata = test_data)
  unname(stats::coef(stats::lm(test_data$y ~ preds))[2])
}
attr(metric_fun, "metric") <- "calibration_slope"


set.seed(123)
est <- simulate_custom(
  data_function = data_fun,
  model_function = model_fun,
  metric_function = metric_fun,
  target_performance = 0.9,
  mean_or_assurance = "assurance",
  min_sample_size = 25,
  max_sample_size = 1000,
  n_reps_total = 1000,
  test_n = 30000,
  progress = FALSE
)
est
est$min_n
plot(est)
```

---

`simulate_survival` *Minimum sample size for survival-outcome prediction models*

---

## Description

Compute the minimum sample size required to develop a prediction model with a **time-to-event (survival)** outcome. As with the other wrappers, this uses a simulation-based learning-curve approach with Gaussian-process surrogate modelling to locate the smallest $n$ meeting the chosen performance criterion.

## Usage

```
simulate_survival(
  signal_parameters,
  noise_parameters = 0,
  complexity = 1,
  data_control = NULL,
  maximum_achievable_cindex,
  baseline_hazard = 1,
  censoring_rate,
  model = c("coxph", "lasso", "ridge", "rf", "xgboost"),
  metric = "calibration_slope",
  target_performance,
  n_reps_total = 1000,
  mean_or_assurance = "assurance",
  ...
)
```

## Arguments

`signal_parameters`
: Integer. Number of candidate predictors associated with the outcome (i.e., true signal features).

`noise_parameters`
: Integer. Number of candidate predictors not associated with the outcome (noise features). Default is 0.

`complexity`
: Integer in 1:4 selecting the data-generating signal structure (see *Data control*). Default 1.

`data_control`
: Optional named list controlling the predictors (see *Data control*). Default `NULL` (generator defaults).

`maximum_achievable_cindex`
: Numeric in (0, 1). Maximum achievable C-index with effectively unlimited data. This is used to calibrate the data-generating mechanism and is not the minimum acceptable threshold.

`baseline_hazard`
: Numeric greater than 0. Baseline hazard level used by the data-generating mechanism (e.g., the constant hazard in an exponential baseline). Larger values imply shorter event times, all else equal.

`censoring_rate`
: Numeric in [0, 1). Proportion of individuals expected to be censored in the simulated datasets (administrative or random censoring). Higher values imply fewer observed events for a fixed $n$.

`model`
: Character string specifying the modelling algorithm. One of `"coxph"` (Cox proportional hazards), `"lasso"`, `"ridge"`, `"rf"` (random survival forest), or `"xgboost"` (gradient boosting with a Cox objective).

`metric`
: Character string naming the performance metric used to assess the sample size; defaults to `"calibration_slope"`. Metric identifiers use one canonical form throughout the package, such as `"calibration_slope"`, `"calibration_in_the_large"`, `"auc"`, `"r2"`, `"cindex"`, and `"csse"`.

"calibration_slope" is the slope from regressing the observed outcome on the model's linear predictor in held-out data; 1 indicates perfect calibration, and values below 1 indicate overfitting. Note that for the machine-learning models ("lasso", "ridge", "rf", "xgboost") this is converted internally to the calibration slope squared error for optimisation and translated back before results are returned; you don't need to do anything, and target_performance is still given on the calibration slope scale. Results derived this way carry a footnote marker in the printed output.

"csse" is the calibration slope squared error, $-(1-s)^2$ for a calibration slope $s$, so that larger is better and 0 is perfect calibration. It can be requested directly, which is mainly useful for advanced use and for comparison against the internal conversion described above. When requesting it directly you are responsible for supplying target_performance on the CSSE scale: a calibration slope target of 0.9 corresponds to a CSSE target of -0.01. No adjustment is applied on your behalf, and results are reported on the CSSE scale.

target_performance

Numeric. Minimum acceptable value of the selected performance metric $M^*$; the algorithm searches for the smallest $n$ meeting the chosen criterion with respect to this threshold.

n_reps_total Integer. Total number of simulation replications used by the engine across the search.

mean_or_assurance

Character string, either "mean" or "assurance". Controls whether the minimum $n$ is defined by the mean-based criterion or the assurance-based criterion (with the assurance level $\delta$ controlled by the engine's defaults or additional arguments in ...).

... Additional options passed to simulate_custom() (e.g., assurance level $\delta$, per-iteration settings).

## Value

An object of class "pmsims" containing the estimated minimum sample size and simulation diagnostics (inputs, fitted GP curve, intermediate evaluations, and summary metrics).

## Criteria

Two formulations are supported.

- **Mean-based**: find the smallest $n$ such that the expected model performance exceeds the target $M^*$, i.e.

$$\min_n \ \mathbb{E}_{D_n}\{M \mid D_n\} \geq M^*.$$

- **Assurance-based**: find the smallest $n$ such that the probability the performance exceeds $M^*$ is at least $\delta$ (e.g. 0.80), i.e.

$$\min_n \ \mathbb{P}_{D_n}(M \mid D_n \geq M^*) \geq \delta.$$

Here, $M$ is the chosen performance metric and the probability/expectation is over repeated samples of training data of size $n$. The assurance criterion explicitly accounts for variability across training sets; models with higher variance typically require larger $n$ to satisfy it.

### Data control

`complexity` selects the signal structure of the data-generating mechanism: `1` purely linear, `2` linear + quadratic, `3` linear + quadratic + interaction, `4` the Friedman function. `data_control` is an optional list fine-tuning the predictors:

`nonlinear_strength` Numeric in `[0, 1)`. Fraction of the signal variance carried by the nonlinear, linearly-inaccessible component. Applies to complexity 2 and 3 only; ignored (with a warning) for 1 and 4. If omitted, the generator's per-complexity default is used.

`correlation` Numeric in $[-1, 1]$. Pairwise correlation among the candidate predictors. Default `0.3`.

`predictor_distribution` One of `"normal"`, `"uniform"`, `"binary"`, `"exponential"`, `"lognormal"`, `"t"`, `"laplace"`. `"binary"` selects 0/1 predictors and requires `binary_predictor_prevalence`; any other value selects continuous predictors from that family. Default `"normal"`.

`binary_predictor_prevalence` Numeric in `(0, 1)`. Prevalence of the binary predictors; required when `predictor_distribution = "binary"`, ignored (with a warning) otherwise. Note: binary predictors are incompatible with complexity 2/3 because squaring a 0/1 variable returns itself.

### See Also

`simulate_binary()`, `simulate_continuous()`, `simulate_custom()`

### Examples

```
set.seed(123)
est <- simulate_survival(
  signal_parameters = 1,
  noise_parameters = 0,
  complexity = 1,
  data_control = list(correlation = 0),
  maximum_achievable_cindex = 0.70,
  baseline_hazard = 0.01,
  censoring_rate = 0.30,
  model = "coxph",
  metric = "calibration_slope",
  target_performance = 0.9,
  mean_or_assurance = "assurance",
  min_sample_size = 25,
  max_sample_size = 500,
  n_reps_total = 1000,
  test_n = 30000,
  progress = FALSE
)
est
est$min_n
plot(est)
```

# Index