

Package: simulab (via r-universe)

October 4, 2026

Title Simulates a Wide Variety of Data Shapes, Sizes and Distributions

Version 0.4.10

Description Generates a wide variety of data, in different shapes, sample sizes, distributions and parameters. The parameters are either drawn at random or set in advance, and in both cases they are stored with the data, so they are known and their recovery can be tested. Generators cover declared variables with treatment assignment and missing values, correlated and clustered data, latent profile, latent class, factor and item response models, longitudinal and multilevel processes, survival times with competing risks, sequences of states, and static, bipartite, multiplex, temporal and transition networks. Every generator returns batches of reproducible data sets for the evaluation of statistical methods, the testing of models and software, and teaching; simulated transition networks are native 'tna' models.

URL <https://pak.dynasite.org/simulab/>,
<https://github.com/mohsaqr/simulab>

BugReports <https://github.com/mohsaqr/simulab/issues>

License MIT + file LICENSE

Encoding UTF-8

Language en-US

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Depends R (>= 4.1.0)

Imports stats, utils

Suggests testthat (>= 3.0.0), knitr, igraph (>= 2.0.0), rmarkdown, simstudy, survival, tna (>= 1.2.3)

Config/testthat/edition 3

VignetteBuilder knitr, rmarkdown

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

Date/Publication 2026-10-04 14:43:52 UTC

RemoteUrl <https://github.com/mohsaqr/simulab>

RemoteRef HEAD

RemoteSha df0e97692099233eb5ca85947794261e9e57f268

Contents

apply_batch	5
apply_conditions	6
as.data.frame.simulab_sim	7
as.data.frame.simulab_tna	8
as_igraph	9
as_tna_model	9
assess_tna_reliability	10
assign_stepped_wedge	11
assign_treatment	13
augment_correlated	14
augment_density	15
augment_factorial	16
augment_markov	17
augment_study	18
augment_survival	19
augment_synthetic	20
block_correlation	21
bootstrap_tna	22
calibrate_distribution	23
calibrate_icc	24
calibrate_logistic	25
calibrate_moments	26
calibrate_survival	27
categorical_formula	28
combine_competing_risks	29
compare_centralities	30
compare_networks	31
compare_tna_models	32
components	33
correlation_structure	33
cross_validate_tna	34
define_condition	35
define_conditions	36
define_missingness	37
define_missingnesses	38
define_survival	39
define_survivals	40
define_variable	41

define_variables	42
encode_factors	44
encode_sequences	45
evaluate_edge_recovery	46
evaluate_tna_estimation	46
expand_clusters	48
expand_periods	49
factorial_design	50
fit_tna	51
fit_tna_batch	52
generate_transition_system	53
global_name_regions	54
global_names	55
inject_missingness	55
learning_state_categories	57
learning_states	57
linear_formula	58
list_distributions	58
list_simulators	59
merge_studies	60
missingness_matrix	60
mixture_formula	62
network centrality	62
observe_treatment	63
observed_data	64
parameter_grid	65
print.simulab_sim	66
print.simulab_tna	67
read_definitions	68
repeat_variables	68
run_simulation_scenario	69
sample_global_names	70
sample_learning_states	71
sample_tna	71
scenario_grid	73
select_variables	73
simulate_anova	74
simulate_bipartite_network	75
simulate_clusters	76
simulate_copula	78
simulate_correlated	80
simulate_correlation	81
simulate_data	82
simulate_density	83
simulate_edge_list	84
simulate_event_log	85
simulate_factors	87
simulate_group_sequences	88

simulate_group_tna	90
simulate_growth	91
simulate_hmm	93
simulate_irt	94
simulate_lca	96
simulate_longitudinal	98
simulate_lpa	100
simulate_markov	101
simulate_ml_lpa	103
simulate_multilevel	105
simulate_multiplex_network	107
simulate_network	108
simulate_network_batches	110
simulate_network_matrix	111
simulate_ordinal	112
simulate_prediction	114
simulate_proportional_survival	115
simulate_regression	117
simulate_scenarios	119
simulate_sequence_batches	120
simulate_sequence_clusters	121
simulate_sequences	122
simulate_spline	124
simulate_study	125
simulate_survival	126
simulate_synthetic	128
simulate_temporal_network	129
simulate_tna	130
simulate_tna_batches	132
simulate_tna_network	133
simulate_ttest	134
simulate_until_event	135
simulation_scenarios	136
spline_basis	137
spline_curves	138
summarize_networks	138
summarize_simulations	139
summarize_transitions	140
summary.simulab_sim	141
survival_curve	141
trim_events	142
update_definition	143
validate_recovery	144
write_simulation	145

apply_batch*Apply a function across simulation results*

Description

Apply a function across simulation results

Usage

```
apply_batch(inputs, fun, ..., id = "batch_id")
```

Arguments

inputs	List with at least one element, commonly data frames or simulation results. Element names label the batches; when inputs is unnamed the positions "1", "2", ... are used instead.
fun	Function applied to each element. When fun has a data argument, as <code>lm()</code> and <code>glm()</code> do, the element is passed as data; otherwise it is the first argument. fun returns a base data.frame, or a fitted model with named coefficients, which becomes a data frame with columns term and estimate. Every call must give the same columns, otherwise the run is an error.
...	Arguments passed to fun.
id	Single string naming the batch-label column, default "batch_id".

Value

A base data.frame stacking the fun outputs by row, with the batch label in a leading column named by id, so one row per row of each output.

Examples

```
inputs <- list(
  small = simulate_ttest(n_a = 20, n_b = 20, mean_a = 0, mean_b = 0.5, seed = 1),
  large = simulate_ttest(n_a = 60, n_b = 60, mean_a = 0, mean_b = 0.5, seed = 2)
)
apply_batch(inputs, fun = summary)

# A model function is applied to each data set and tidied to coefficients.
datasets <- simulate_regression(seed = 1, batch = 3)
apply_batch(datasets, lm, formula = outcome ~ x1 + x2)
```

apply_conditions	<i>Apply conditional generation rules</i>
------------------	---

Description

Rules are evaluated in order, so later rules may intentionally replace values written by earlier rules. A row no rule selects is left missing.

Usage

```
apply_conditions(data, specification, seed = NULL, envir = parent.frame())
```

Arguments

data	Input base data.frame.
specification	Rules from define_conditions() , in either the rule form written with when() or the formula/variance column form.
seed	Optional random seed.
envir	Environment used to resolve values a condition or formula names but the data does not carry. Distribution parameters inside when() are evaluated against the data alone, so they may only name columns.

Value

A simulab_sim base data.frame with conditional variables added.

Examples

```
data <- data.frame(id = 1:100, group = rep(0:1, each = 50))

# Rules written with when() may use any distribution in the catalogue.
result <- apply_conditions(
  data,
  specification = define_conditions(
    outcome = when(group == 1, normal(mean = 5, sd = 1)),
    outcome = when(group == 0, poisson(lambda = 2))
  ),
  seed = 1
)
head(result)

specification <- define_conditions(
  define_condition("outcome", condition = "group == 1",
    formula = "2", variance = "1", distribution = "normal"),
  define_condition("outcome", condition = "group == 0",
    formula = "0", variance = "1", distribution = "normal")
)
result <- apply_conditions(data, specification = specification, seed = 1)
head(result)
```

`as.data.frame.simulab_sim`*Convert a simulation component to a tidy data frame*

Description

Convert a simulation component to a tidy data frame

Usage

```
## S3 method for class 'simulab_sim'  
as.data.frame(x, row.names = NULL, optional = FALSE, what = "data", ...)
```

Arguments

<code>x</code>	A <code>simulab_sim</code> object.
<code>row.names</code>	Ignored.
<code>optional</code>	Ignored.
<code>what</code>	Single string naming the table to return, default "data" for the primary simulated observations. Use <code>components(x)</code> to list the available choices; an unrecognized name is an error that lists them.
<code>...</code>	Reserved for future methods.

Value

A plain base `data.frame` (the `simulab_sim` class and attributes are dropped) containing the requested simulation table.

Examples

```
result <- simulate_ttest(  
  n_a = 10,  
  n_b = 10,  
  mean_a = 0,  
  mean_b = 0.5,  
  seed = 1  
)  
as.data.frame(result)  
as.data.frame(result, what = "parameters")
```

as.data.frame.simulab_tna

Tidy tables of a simulated transition network

Description

Tidy tables of a simulated transition network

Usage

```
## S3 method for class 'simulab_tna'
as.data.frame(x, row.names = NULL, optional = FALSE, what = "edges", ...)
```

Arguments

x	A simulab_tna result of <code>simulate_tna()</code> .
row.names, optional	Ignored; present for the generic.
what	Table to return: "edges" (the default, the fitted network), "transitions", "initial_probabilities", "sequences", "wide" or "model_info".
...	Ignored.

Value

A base data.frame. "edges" has one row per edge with columns from, to and weight; "transitions" one row per edge with columns from, to and probability; "initial_probabilities" one row per state with columns state and probability; "sequences" one row per sequence and position with columns id, period and state.

Examples

```
if (requireNamespace("tna", quietly = TRUE)) {
  network <- simulate_tna(n = 30, n_states = 3, seed = 1)
  as.data.frame(network)
  as.data.frame(network, what = "transitions")
}
```

as_igraph

*Convert tidy network data to an igraph object***Description**

Convert tidy network data to an igraph object

Usage

```
as_igraph(x, directed = NULL)
```

Arguments

x	A simulab network, tidy edge list, matrix, or native TNA model.
directed	Whether the resulting graph is directed. NULL (the default) takes it from the settings table of a simulab network, so a network generated with directed = FALSE converts to an undirected graph, and treats any other input as directed.

Value

A native igraph object carrying a weight edge attribute. When x is a simulab_sim with a nodes component, that node set supplies the vertices, so isolates are preserved; otherwise the vertices are the nodes appearing in the edge list.

Examples

```
network <- simulate_network(nodes = 30, model = "bernoulli", probability = 0.1, seed = 1)
if (requireNamespace("igraph", quietly = TRUE)) {
  graph <- as_igraph(network)
  igraph::vcount(graph)
}
```

as_tna_model

*Convert a tidy simulab TNA result to a native tna model***Description**

Convert a tidy simulab TNA result to a native tna model

Usage

```
as_tna_model(x, group = NULL)
```

Arguments

x	A result from <code>fit_tna()</code> or <code>simulate_group_tna()</code> .
group	Optional group name for a grouped model. When omitted, the complete native grouped model is returned.

Value

A native `tna` or `group_tna` model object. For a `compare_tna_models()` result the stored models are returned as a list named by model type instead.

Examples

```
data <- simulate_sequences(n = 40, n_states = 3, chain_length = 12, seed = 1)
if (requireNamespace("tna", quietly = TRUE)) {
  model <- as_tna_model(fit_tna(data, model = "tna"))
  class(model)
}
```

`assess_tna_reliability`

Assess split-half TNA reliability

Description

Splits the sequences at random into two halves, fits the same estimator to each, and compares the two networks. Repeating that over iterations independent splits shows how stable an estimate this much data supports.

Usage

```
assess_tna_reliability(
  data,
  model = c("tna", "ftna", "ctna", "atna"),
  iterations = 100L,
  split = 0.5,
  format = c("auto", "long", "wide"),
  id = "id",
  period = "period",
  state = "state",
  seed = NULL,
  ...
)
```

Arguments

<code>data</code>	Sequence data accepted by <code>fit_tna()</code> , with at least 4 rows.
<code>model</code>	TNA estimator, one of "tna" (the default), "ftna", "ctna" or "atna".
<code>iterations</code>	Number of random split halves. A single whole number of at least 2, defaulting to 100.
<code>split</code>	Fraction of the sequences assigned to the first half, rounded down. A single number strictly between 0 and 1, defaulting to 0.5. Each half must end up with at least two sequences.
<code>format, id, period, state</code>	Input-format arguments passed on when the sequences are prepared: <code>format</code> is one of "auto" (the default), "long" or "wide", and the other three name the identifier, period and state columns of long input.
<code>seed</code>	Optional seed. A single number, or NULL (the default).
<code>...</code>	Arguments passed to <code>fit_tna()</code> .

Value

A plain base `data.frame`, not a `simulab_sim`, with one row per split, holding iteration, model and the `compare_networks()` agreement metrics for the two halves.

Examples

```
data <- simulate_sequences(n = 40, n_states = 3, chain_length = 12, seed = 1)
if (requireNamespace("tna", quietly = TRUE)) {
  assess_tna_reliability(data, model = "tna", iterations = 2, seed = 1)
}
```

`assign_stepped_wedge` *Assign a stepped-wedge treatment schedule*

Description

Splits the clusters into equally sized waves and switches each wave on at `start + (wave - 1) * wave_length`, the wave's treatment-start period. A row is treated from `treatment_start + lag` onwards, and the lag periods from `treatment_start` up to that point are flagged as transition periods instead. The period values are compared as they stand, so they must be on the same scale as `start`. Periods created by `expand_periods()` are numbered from 0.

Usage

```
assign_stepped_wedge(
  data,
  cluster,
  period,
  waves,
```

```

    wave_length,
    start,
    lag = 0L,
    treatment = "treatment",
    transition = "transition",
    randomize = TRUE,
    seed = NULL
  )

```

Arguments

data	Long-form base data.frame, or a simulab_sim, with one row per cluster unit and period and at least one row.
cluster	Cluster identifier. A single string naming a column of data, whose number of unique values must be divisible by waves.
period	Period variable. A single string naming a column of data, which must reach $\text{start} + (\text{waves} - 1) * \text{wave_length} + \text{lag}$.
waves	Number of waves. A single positive whole number.
wave_length	Periods between consecutive wave starts. A single positive whole number.
start	Treatment-start period of the first wave. A single number.
lag	Transition periods after a cluster's start before treatment becomes active. A single non-negative whole number, defaulting to 0.
treatment	Name of the active-treatment variable, which is 1 from $\text{treatment_start} + \text{lag}$ onwards and 0 before. A single string, defaulting to "treatment".
transition	Name of the transition variable, which is 1 in the lag periods between a cluster's treatment_start and the start of active treatment. A single string, defaulting to "transition". The column is only added when lag is above 0.
randomize	Randomize the allocation of clusters to waves. A single flag, defaulting to TRUE. FALSE fills the waves in the order the clusters appear in data.
seed	Optional random seed. A single number, or NULL (the default).

Value

A simulab_sim base data.frame with the columns of data followed by the transition variable (only when lag is above 0) and the treatment variable, one row per input row. The component schedule, reached with `as.data.frame(x, what = "schedule")`, holds one row per cluster with the cluster identifier under its own name plus columns wave and treatment_start.

Examples

```

data <- expand_clusters(
  data.frame(cluster = 1:6), cluster = "cluster", size = 4
)
data <- expand_periods(data, periods = 5, id = "id", period = "period")

result <- assign_stepped_wedge(
  data, cluster = "cluster", period = "period",

```

```

    waves = 3, wave_length = 1, start = 2, seed = 1
  )
  head(result)

```

assign_treatment	<i>Assign randomized treatment groups</i>
------------------	---

Description

Randomizes the rows of data to treatment groups, independently within each combination of the strata variables. Balanced assignment allocates the group sizes a stratum is due, rounding the shares to whole units and giving the remainder to the groups with the largest fractional part, then permutes them; unbalanced assignment draws each row independently with probabilities proportional to ratios.

Usage

```

assign_treatment(
  data,
  groups = 2L,
  balanced = TRUE,
  strata = NULL,
  ratios = NULL,
  name = "treatment",
  seed = NULL
)

```

Arguments

data	Base data.frame, or a simulab_sim, with at least one row.
groups	Number of treatment groups, as a single whole number of at least 2 (defaulting to 2), or an atomic vector of at least two unique labels. A count of 2 is labeled 0 and 1, and a larger count 1 to groups.
balanced	Use exact allocation within each stratum. A single flag, defaulting to TRUE. FALSE draws each row independently instead, so the realized group sizes vary.
strata	Optional character vector of stratification variables, all present in data. Defaults to NULL, one stratum holding every row.
ratios	Relative allocation ratios, a positive numeric vector with one value per group. Defaults to NULL, equal allocation.
name	Name of the treatment variable, which must not already exist in data. A single non-empty string, defaulting to "treatment".
seed	Optional random seed. A single number, or NULL (the default).

Value

A `simulab_sim` base `data.frame` with the columns of data followed by the treatment variable named by name, one row per input row. The component `allocation`, reached with `as.data.frame(x, what = "allocation")`, holds one row per stratum and group, with columns `stratum`, `treatment` and `observations`.

Examples

```
data <- data.frame(id = 1:100, site = rep(c("north", "south"), each = 50))

result <- assign_treatment(data, groups = 2, strata = "site", seed = 1)
head(result)
as.data.frame(result, what = "allocation")
```

<code>augment_correlated</code>	<i>Add correlated variables to existing data</i>
---------------------------------	--

Description

Add correlated variables to existing data

Usage

```
augment_correlated(
  data,
  specification,
  rho = 0,
  tau = NULL,
  structure = c("independent", "exchangeable", "ar1", "custom"),
  correlation = NULL,
  group = NULL,
  seed = NULL,
  envir = parent.frame()
)
```

Arguments

<code>data</code>	Base <code>data.frame</code> .
<code>specification</code>	New variable definitions, in the formula/variance column form built by <code>define_variable()</code> . Unlike <code>simulate_copula()</code> , <code>augment_correlated()</code> does not accept the distribution-call form, and one row of <code>specification</code> is one new variable. None of the named variables may already exist in <code>data</code> .
<code>rho</code> , <code>tau</code> , <code>structure</code> , <code>correlation</code>	Copula correlation arguments. <code>structure</code> follows the same resolution rule as <code>simulate_copula()</code> : leaving it unset selects "exchangeable" when a non-zero <code>rho</code> or <code>tau</code> is given. <code>tau</code> is converted to the latent Pearson correlation $\sin(\pi * \tau / 2)$.

group	Optional grouping variable. With grouping, one definition is generated as correlated observations within each group.
seed	Optional random seed.
envir	Formula evaluation environment.

Value

A `simulab_sim` base data.frame holding every column of data plus one column per new variable. Two tidy tables come from `as.data.frame()`: `what = "definitions"` restates the specification, and `what = "latent_correlation"` describes the latent Gaussian correlation. Without group that table has one row per correlation-matrix cell, with columns `row`, `column` and `correlation`; with group the correlation is across the observations inside each group, so the table instead has one row per group with columns `group`, `observations`, `rho` and `structure`.

Examples

```
data <- data.frame(id = 1:100)
specification <- define_variables(
  define_variable("a", formula = "0", variance = "1", distribution = "normal"),
  define_variable("b", formula = "0", variance = "1", distribution = "normal")
)
result <- augment_correlated(data, specification = specification, rho = 0.5, seed = 1)
head(result)
```

augment_density	<i>Add an empirical-density variable to existing data</i>
-----------------	---

Description

Add an empirical-density variable to existing data

Usage

```
augment_density(
  data,
  values,
  variable,
  use_limits = FALSE,
  keep_missing = FALSE,
  seed = NULL
)
```

Arguments

data	Destination base data.frame.
values	Numeric source values, of which at least two must be present and distinct.
variable	Name of the new variable.
use_limits, keep_missing, seed	Density-simulation arguments, as in simulate_density() .

Value

A `simulab_sim` base `data.frame` with the density variable added.

Examples

```
values <- stats::rnorm(200, mean = 5, sd = 2)
data <- data.frame(id = 1:50)
result <- augment_density(data, values = values, variable = "score", seed = 1)
head(result)
```

<code>augment_factorial</code>	<i>Add factorial conditions to existing rows</i>
--------------------------------	--

Description

Add factorial conditions to existing rows

Usage

```
augment_factorial(data, factors, coding = c("dummy", "effect", "level"))
```

Arguments

<code>data</code>	Input base <code>data.frame</code> , with at least one row.
<code>factors</code>	Named numeric vector of whole numbers giving the number of levels of each factor. Every entry must be at least 2.
<code>coding</code>	Factor coding passed to <code>factorial_design()</code> : "dummy" (the default), "effect" or "level".

Value

A `simulab_sim` base `data.frame` with one row per row of data crossed with every design condition, so `nrow(data) * prod(factors)` rows: the columns of data followed by one column per factor. The design component holds the distinct conditions, one row each.

Examples

```
data <- data.frame(id = 1:12)
result <- augment_factorial(data, factors = c(dose = 3L, timing = 2L))
head(result)
```

augment_markov	<i>Add a Markov chain to each input row</i>
----------------	---

Description

Add a Markov chain to each input row

Usage

```
augment_markov(
  data,
  transition,
  chain_length,
  initial = NULL,
  states = NULL,
  trim_state = NULL,
  id = "id",
  period = "period",
  state = "state",
  seed = NULL
)
```

Arguments

data	Base data.frame with a unique identifier.
transition, chain_length, states, trim_state	Markov-chain arguments.
initial	Starting-state probabilities or the name of an input variable containing each row's starting state. When NULL, every chain starts in the first state.
id, period, state	Output variable names.
seed	Optional random seed.

Value

A long-form simulab_sim base data.frame combining the input variables with one row per chain position: id, period, state, followed by the other input columns. A transitions table (from, to, probability) is available through as.data.frame().

Examples

```
data <- data.frame(id = 1:20, group = rep(c("a", "b"), each = 10))

result <- augment_markov(
  data,
  transition = matrix(c(0.7, 0.3, 0.4, 0.6), nrow = 2, byrow = TRUE),
  chain_length = 10,
```

```

    states = c("A", "B"),
    seed = 1
  )
  head(result)

```

augment_study

Add declaratively specified variables to existing data

Description

A parameter or formula may refer to a column data already has as readily as to a variable defined earlier in the same specification. A variable that already exists is refused rather than overwritten.

Usage

```
augment_study(data, specification, seed = NULL, envir = parent.frame())
```

Arguments

data	A base data.frame.
specification	A tidy definition table created with <code>define_variables()</code> , in either the distribution-call form or the formula/variance column form.
seed	Optional random seed.
envir	Environment used to resolve functions and external values named by a formula/variance specification. It is not consulted by the distribution-call form, whose parameters must be expressions over columns of data or variables defined earlier in the same specification.

Value

A `simulab_sim` base data.frame with one row per row of data, containing the original and new variables. Use `as.data.frame(x, what = "definitions")` for the generating definitions.

Conditions

`simulab_existing_variable` when the specification names a column data already has.

Examples

```

# Distribution calls, generated into data that is already there.
head(augment_study(
  data.frame(id = 1:100, treated = rep(0:1, each = 50)),
  specification = define_variables(
    outcome = normal(mean = 3 + 2 * treated, sd = 1),
    visits = poisson(lambda = 4)
  ),
  seed = 3
))

```

```

base <- simulate_study(
  n = 100,
  specification = define_variables(
    define_variable("baseline", formula = "0", variance = "1", distribution = "normal")
  ),
  seed = 1
)

result <- augment_study(
  base,
  specification = define_variables(
    define_variable("outcome", formula = "2 * baseline", variance = "1",
      distribution = "normal")
  ),
  seed = 2
)
head(result)

```

augment_survival	<i>Add one or more survival processes to data</i>
------------------	---

Description

Draws one latent event time per process for every row of data, from the Weibull hazard of [define_survival\(\)](#). The times are uncensored: censoring and competing-risk coding are applied afterwards, for example with [combine_competing_risks\(\)](#). An event with several rows in specification has a piece-wise hazard, whose scale and shape must stay constant across its segments.

Usage

```

augment_survival(
  data,
  specification,
  seed = NULL,
  digits = NULL,
  envir = parent.frame()
)

```

Arguments

data	Baseline base data.frame, or a simulab_sim, with at least one row. Its columns are the covariates the formulas may refer to, and none of them may already carry an event name.
specification	Definitions from define_survivals() , a simulab_survival_spec object.
seed	Optional random seed. A single number, or NULL (the default) to leave the stream untouched.

digits	Optional number of decimal places the event times are rounded to. A single non-negative whole number, or NULL (the default) for unrounded times.
envir	Environment the formulas are evaluated in after the data columns. Defaults to the caller's environment.

Value

A `simulab_sim` base `data.frame` with the columns of data followed by one numeric event-time column per process, named by its event, and one row per input row. The component `survival_definitions`, reached with `as.data.frame(x, what = "survival_definitions")`, is the specification as a plain `data.frame`.

Examples

```
data <- data.frame(id = 1:100, treatment = rep(0:1, each = 50))

result <- augment_survival(
  data,
  specification = define_survivals(
    define_survival("time", formula = "-8 + 0.5 * treatment", shape = 0.3)
  ),
  seed = 1
)
head(result)
```

augment_synthetic	<i>Add jointly resampled source variables to existing data</i>
-------------------	--

Description

Add jointly resampled source variables to existing data

Usage

```
augment_synthetic(data, source, variables = NULL, seed = NULL)
```

Arguments

data	Destination base <code>data.frame</code> .
source	Source base <code>data.frame</code> .
variables	Variables to resample. NULL, the default, resamples every variable in source.
seed	Optional random seed.

Value

A `simulab_sim` base `data.frame` with the resampled variables added, one source row drawn per row of data. `as.data.frame(x, what = "provenance")` gives the source row each destination row was drawn from.

Examples

```
source_data <- data.frame(x = stats::rnorm(100))
data <- data.frame(id = 1:20)
result <- augment_synthetic(data, source = source_data, variables = "x", seed = 1)
head(result)
```

block_correlation	Create clustered-period correlation structures
-------------------	--

Description

Create clustered-period correlation structures

Usage

```
block_correlation(
  n_individuals,
  n_periods,
  within_period,
  between_period = 0,
  within_individual = NULL,
  decay = NULL,
  design = c("cross_sectional", "cohort"),
  n_clusters = 1L
)
```

Arguments

n_individuals	Individuals per period. Supply one value, one value per cluster, or one value per cluster-period combination.
n_periods	Number of periods.
within_period	Correlation between different individuals in the same period.
between_period	Correlation between different individuals in different periods for an exchangeable structure.
within_individual	Correlation for the same individual across periods in a cohort design.
decay	Optional period-to-period correlation decay. Supplying this selects a decay structure.
design	Cross-sectional or closed-cohort design.
n_clusters	Number of cluster-specific structures.

Value

A base data.frame with one row per cluster and correlation-matrix cell, and columns row, column, correlation and cluster. The row and column labels are of the form period_<p>_unit_<u>.

Examples

```
block_correlation(
  n_individuals = 3,
  n_periods = 4,
  within_period = 0.3,
  between_period = 0.1,
  design = "cross_sectional"
)
```

bootstrap_tna

*Bootstrap a TNA model***Description**

Bootstrap a TNA model

Usage

```
bootstrap_tna(
  data,
  model = c("tna", "ftna", "ctna", "atna"),
  repetitions = 100L,
  fraction = 1,
  format = c("auto", "long", "wide"),
  id = "id",
  period = "period",
  state = "state",
  group = NULL,
  seed = NULL,
  ...
)
```

Arguments

data	Sequence data in long or wide form.
model	TNA estimator, one of "tna" (the default), "ftna", "ctna", "atna".
repetitions	Single whole number of bootstrap samples, at least 2, default 100.
fraction	Single number in (0, 1] giving the fraction of sequences drawn with replacement in each repetition, default 1. At least two sequences are always drawn.
format, id, period, state, group	Arguments describing the shape and column names of data. They are used to reshape data before resampling; the resampled sequences are then fitted in wide form, so only group is forwarded to fit_tna() . group resamples within each group.
seed	Optional single seed, restored on exit and recorded on the result.
...	Further arguments passed to fit_tna() .

Value

A `simulab_sim` edge list with one row per repetition and transition: a leading iteration column followed by the columns `fit_tna()` returns. The summary component holds one row per edge (per group, when group is given) with the bootstrap mean, sd, and the 2.5% and 97.5% percentiles as lower and upper.

Examples

```
data <- simulate_sequences(n = 40, n_states = 3, chain_length = 12, seed = 1)
if (requireNamespace("tna", quietly = TRUE)) {
  head(bootstrap_tna(data, model = "tna", repetitions = 5, seed = 1))
}
```

`calibrate_distribution`

Convert mean/dispersion parameterizations

Description

`calibrate_distribution()` inverts the mean and dispersion parameterization used by the formula/variance specification columns into the shape parameters base R's samplers take. For a mean and a *variance*, and for the wider catalogue, use `calibrate_moments()`.

Usage

```
calibrate_distribution(
  distribution = c("beta", "gamma", "negative_binomial"),
  mean,
  dispersion
)
```

Arguments

<code>distribution</code>	Beta, gamma, or negative-binomial distribution.
<code>mean</code>	Distribution mean.
<code>dispersion</code>	Beta precision or gamma/negative-binomial dispersion.

Value

A base data.frame with one row per input value and parameter, and columns `distribution`, `mean`, `dispersion`, `parameter` and `value`.

Examples

```
calibrate_distribution("beta", mean = 0.4, dispersion = 8)
calibrate_distribution("gamma", mean = 10, dispersion = 0.5)
calibrate_distribution("negative_binomial", mean = 5, dispersion = 0.4)
```

calibrate_icc

*Calibrate random-effect variance for a target ICC***Description**

Calibrate random-effect variance for a target ICC

Usage

```
calibrate_icc(
  icc,
  distribution = c("normal", "binary", "poisson", "gamma", "negative_binomial"),
  total_variance = NULL,
  within_variance = NULL,
  mean = NULL,
  dispersion = NULL
)
```

Arguments

icc	Target intraclass correlations.
distribution	Outcome distribution.
total_variance	Total normal-outcome variance.
within_variance	Within-cluster normal-outcome variance.
mean	Poisson or negative-binomial mean.
dispersion	Gamma or negative-binomial dispersion.

Value

A base data.frame with one row per target ICC and columns icc, distribution and random_effect_variance. Except for the normal total_variance case, which returns $icc * total_variance$, the variance is $icc / (1 - icc)$ times a residual variance on the linear-predictor scale: within_variance for normal, the logistic latent variance $\pi^2 / 3$ for binary, $\log(1 + 1 / mean)$ for poisson, $\text{trigamma}(1 / dispersion)$ for gamma, and $\text{trigamma}(1 / (1 / mean + dispersion))$ for negative_binomial.

Examples

```
calibrate_icc(icc = 0.1, distribution = "normal", total_variance = 1)
calibrate_icc(icc = c(0.05, 0.1), distribution = "binary")
```

calibrate_logistic	<i>Calibrate logistic coefficients to prevalence and treatment contrast</i>
--------------------	---

Description

Calibrate logistic coefficients to prevalence and treatment contrast

Usage

```
calibrate_logistic(
  data,
  coefficients,
  prevalence,
  risk_ratio = NULL,
  risk_difference = NULL,
  treatment = "treatment",
  treatment_prevalence = 0.5,
  auc = NULL,
  tolerance = 1e-08
)
```

Arguments

data	Covariate data.
coefficients	Named covariate coefficients.
prevalence	Target population prevalence.
risk_ratio	Optional treated/control risk ratio.
risk_difference	Optional treated-control risk difference.
treatment	Name for the optional treatment coefficient.
treatment_prevalence	Treatment prevalence used for calibration.
auc	Optional target population area under the ROC curve. Covariate coefficients are proportionally scaled to reach this target.
tolerance	Numerical root tolerance.

Value

A base data.frame with one row per calibrated coefficient and columns term, coefficient, calibration and coefficient_scale. Coefficients are log-odds on the logit scale. The first row is "(Intercept)", followed by the treatment row when risk_ratio or risk_difference was supplied, then one row per name of coefficients. calibration names the target that was solved for – "prevalence", "auc", "risk_ratio" or "risk_difference" – and coefficient_scale is the factor the supplied covariate coefficients were multiplied by, which is 1 unless auc was given.

Examples

```
set.seed(1)
covariates <- data.frame(x1 = stats::rnorm(200), x2 = stats::rnorm(200))

calibrate_logistic(
  covariates, coefficients = c(x1 = 0.5, x2 = -0.3), prevalence = 0.2
)

# Scale the coefficients to hit a target discrimination instead.
calibrate_logistic(
  covariates, coefficients = c(x1 = 0.5, x2 = -0.3),
  prevalence = 0.2, auc = 0.75
)
```

calibrate_moments	<i>Solve a distribution's parameters from a target mean and variance</i>
-------------------	--

Description

`calibrate_moments()` inverts the moments of a distribution into the parameters that attain them, so a target mean and variance can be stated directly and turned into a distribution call. A one-parameter family takes mean alone, because its mean already fixes its variance; the reported variance is then the one those parameters imply.

Usage

```
calibrate_moments(distribution = NULL, mean = NULL, variance = NULL)
```

Arguments

distribution	Distribution name. <code>calibrate_moments()</code> with no arguments reports the distributions it can invert.
mean	Target mean, or a vector of target means.
variance	Target variance, for a two-parameter family. Omitted for a one-parameter family, whose variance its mean determines.

Details

Most inversions are closed form. For a scale family whose shape is fixed by the coefficient of variation alone – Weibull, log-logistic, Frechet, Nakagami – the shape is found by root finding and the scale then follows exactly. Two one-parameter families, chi and zero_truncated_poisson, likewise invert their mean by root finding. Some families cannot reach every pair: a Nakagami squared coefficient of variation is at most $\pi / 2 - 1$, a Lomax with a finite variance always has one above 1, and a beta variance is below $\text{mean} * (1 - \text{mean})$. Those are refused rather than approximated.

Value

A base data.frame with one row per target and parameter, and columns distribution, mean, variance, parameter and value. The parameter names are those `list_distributions()` reports, so the result states a distribution call. For a one-parameter family the reported variance is the one the solved parameters imply rather than a target. Called with no arguments the result is instead the catalogue: one row per invertible distribution, with columns distribution, parameters and targets.

Conditions

`simulab_unattainable_moments` when no member of the family has the requested moments, such as a beta variance at or above $\text{mean} * (1 - \text{mean})$; `simulab_no_moment_solution` when the distribution has no implemented inversion; `simulab_incompatible_lengths` when mean and variance are vectors of different, non-recyclable lengths.

Examples

```
# The 36 distributions that can be inverted, and the targets each takes.
calibrate_moments()

# A lognormal with mean 10 and variance 25.
calibrate_moments("lognormal", mean = 10, variance = 25)

# A one-parameter family takes the mean alone and reports the variance
# that follows from it.
calibrate_moments("poisson", mean = 4)

# Targets recycle, so a sweep is one call.
calibrate_moments("gamma", mean = c(5, 10, 20), variance = 9)
```

calibrate_survival	<i>Calibrate a Weibull survival curve to target points</i>
--------------------	--

Description

Finds the formula and shape whose `simstudy` Weibull curve, $S(t) = \exp(-\exp(\text{formula}) * t^{(1 / \text{shape})})$ with scale fixed at 1, passes as closely as possible through the supplied points. The fit is a least-squares fit of $\log(\text{time})$ on $\log(-\log(\text{survival}))$, run with L-BFGS-B, and the returned formula and shape can be handed straight to `define_survival()` or `survival_curve()`.

Usage

```
calibrate_survival(time, survival)
```

Arguments

time	Increasing positive times. A finite numeric vector of at least two strictly increasing values, the same length as survival.
survival	Survival probabilities at those times. A finite numeric vector strictly inside (0, 1) and strictly decreasing.

Value

A one-row base data.frame with the calibrated numeric formula and shape, the `optim()` convergence code (always 0, since a non-converged fit raises an error), and `rmse`, the root mean squared residual **on the log-time scale**.

Examples

```
calibrate_survival(time = c(50, 100, 150), survival = c(0.9, 0.7, 0.4))
```

categorical_formula	<i>Construct a categorical probability formula</i>
---------------------	--

Description

Supply exactly one of probabilities and categories; supplying both, or neither, is an error. Probabilities are normalized to sum to one.

Usage

```
categorical_formula(probabilities = NULL, categories = NULL)
```

Arguments

probabilities	Category probabilities, at least two of them.
categories	Number of equal-probability categories, used instead of probabilities.

Value

A one-row `simulab_formula` base data.frame.

Examples

```
categorical_formula(probabilities = c(0.2, 0.5, 0.3))
```

 combine_competing_risks

Combine competing event times

Description

Reduces several latent event times to the one observed first: the observed time is their row-wise minimum and the event code says which process won. Naming one of them in `censor` turns that process into censoring, so its code is 0 and the remaining processes are coded 1, 2, ... in the order they appear in events. Without `censor` every row is an event and no code is 0. Ties are broken in favor of the earlier entry of events.

Usage

```
combine_competing_risks(
  data,
  events,
  censor = NULL,
  time = "time",
  event = "event",
  type = "event_type",
  keep_events = FALSE
)
```

Arguments

<code>data</code>	Base <code>data.frame</code> , or a <code>simulab_sim</code> , containing the event-time variables, with at least one row.
<code>events</code>	Event-time variable names. A character vector of at least two names, all present in <code>data</code> , holding complete numeric times.
<code>censor</code>	Optional name of the one entry of events that represents censoring rather than an event. A single string, or <code>NULL</code> (the default) when every process is an event.
<code>time</code>	Name of the observed-time variable. A single string, defaulting to <code>"time"</code> .
<code>event</code>	Name of the integer event-code variable. A single string, defaulting to <code>"event"</code> . The code is 0 for the process named in <code>censor</code> and 1, 2, ... for the other entries of events, in their given order.
<code>type</code>	Name of the character event-type variable, holding the winning process name. A single string, defaulting to <code>"event_type"</code> .
<code>keep_events</code>	Retain the component event-time variables. A single flag, defaulting to <code>FALSE</code> , which drops them.

Value

A `simulab_sim` base `data.frame` with one row per input row: the columns of `data` (without the entries of events unless `keep_events` is `TRUE`) followed by `time`, `event` and `type` under the

names given by those arguments. The component events, reached with `as.data.frame(x, what = "events")`, is the codebook, one row per process with columns `event_type`, `event_code` and the logical censoring.

Examples

```
data <- data.frame(id = 1:100, treatment = rep(0:1, each = 50))
data <- augment_survival(
  data,
  specification = define_survivals(
    define_survival("time_relapse", formula = -8, shape = 0.3),
    define_survival("time_death", formula = -9, shape = 0.3)
  ),
  seed = 1
)

result <- combine_competing_risks(
  data, events = c("time_relapse", "time_death")
)
head(result)
```

compare_centralities	<i>Compare node centralities between two networks</i>
----------------------	---

Description

Compare node centralities between two networks

Usage

```
compare_centralities(
  x,
  y,
  measures = c("degree", "betweenness", "closeness"),
  method = c("pearson", "spearman", "kendall"),
  directed = NULL
)
```

Arguments

<code>x, y</code>	Supported network representations.
<code>measures</code>	Centralities passed to <code>network centrality()</code> .
<code>method</code>	Correlation method.
<code>directed</code>	Treat networks as directed. NULL (the default) takes it from each network's settings table.

Value

A tidy base data.frame with one row per measure and columns measure, method, correlation, mae, and nodes. Nodes present in only one network contribute a centrality of zero to the other. correlation is NA when either set of centralities has no variance. Requires the suggested igraph package.

Examples

```
a <- simulate_network(nodes = 30, model = "bernoulli", probability = 0.1, seed = 1)
b <- simulate_network(nodes = 30, model = "bernoulli", probability = 0.1, seed = 2)
if (requireNamespace("igraph", quietly = TRUE)) {
  compare_centralities(a, b, measures = "degree")
}
```

compare_networks

Compare two weighted networks

Description

Compare two weighted networks

Usage

```
compare_networks(x, y, threshold = 0)
```

Arguments

x, y	Supported network representations.
threshold	Absolute weight threshold for edge presence: a dyad counts as an edge when $\text{abs}(\text{weight}) > \text{threshold}$.

Details

The two networks are aligned on the union of the dyads that carry an edge in either network; dyads absent from both are excluded, so every metric is conditional on that union rather than on all possible dyads.

Value

A one-row base data.frame with columns pearson, cosine, mae, rmse, jaccard, edges_x, and edges_y. pearson and cosine are NA when either network's aligned weights have no variance or are all zero.

Examples

```
a <- simulate_network(nodes = 30, model = "bernoulli", probability = 0.1, seed = 1)
b <- simulate_network(nodes = 30, model = "bernoulli", probability = 0.1, seed = 2)
compare_networks(a, b)
```

compare_tna_models	<i>Compare multiple TNA estimators on the same sequences</i>
--------------------	--

Description

Compare multiple TNA estimators on the same sequences

Usage

```
compare_tna_models(
  data,
  models = c("tna", "ftna", "ctna", "atna"),
  format = c("auto", "long", "wide"),
  id = "id",
  period = "period",
  state = "state",
  group = NULL,
  ...
)
```

Arguments

data	Sequence data accepted by <code>fit_tna()</code> .
models	One or more TNA model types.
format, id, period, state, group	Input arguments passed to <code>fit_tna()</code> .
...	Estimator arguments.

Value

A tidy edge-list `simulab_sim` base `data.frame` with one row per model/group/edge and columns `model`, `from`, `to`, and `weight` (with `group` inserted when `group` is supplied). A stacked `model_info` table, one row per fitted model and group, is available through `as.data.frame()`. The fitted native models are stored as a named list and returned by `as_tna_model()`.

Examples

```
data <- simulate_sequences(n = 40, n_states = 3, chain_length = 12, seed = 1)
if (requireNamespace("tna", quietly = TRUE)) {
  compare_tna_models(data, models = c("tna", "ftna"))
}
```

components

List the tidy tables available from a simulation

Description

List the tidy tables available from a simulation

Usage

```
components(x)
```

Arguments

x A simulab_sim object.

Value

A base data.frame with one row per available table and the columns table (the name to pass as what to as.data.frame()), rows and columns. The first row is always the primary "data" table; a result that carries no extra tables returns that single row.

Examples

```
result <- simulate_ttest(
  n_a = 10,
  n_b = 10,
  mean_a = 0,
  mean_b = 0.5,
  seed = 1
)
components(result)
```

correlation_structure *Create a tidy correlation structure*

Description

Create a tidy correlation structure

Usage

```
correlation_structure(
  n_variables,
  rho = 0,
  structure = c("independent", "exchangeable", "ar1", "custom"),
  correlation = NULL,
  variable_names = NULL
)
```

Arguments

n_variables	Number of variables.
rho	Correlation coefficient used by exchangeable and AR(1) structures.
structure	Correlation structure: one of "independent", "exchangeable", "ar1", or "custom". If left unset it is chosen from the other arguments: "custom" when correlation is supplied, "exchangeable" when a non-zero rho is supplied, and "independent" otherwise. Passing "independent" together with a non-zero rho is a contradiction and raises a <code>simulab_contradictory_structure</code> error.
correlation	Custom correlation matrix.
variable_names	Optional variable names.

Value

A base `data.frame` with one row per matrix cell and columns `row`, `column`, and `correlation`.

Examples

```
correlation_structure(
  n_variables = 3,
  rho = 0.5,
  structure = "ar1"
)
```

cross_validate_tna	<i>Cross-validate TNA estimators</i>
--------------------	--------------------------------------

Description

Each iteration splits the sequences once into a training and a testing set, fits every estimator to both halves, and compares the two networks with `compare_networks()`.

Usage

```
cross_validate_tna(
  data,
  models = c("tna", "ftna", "ctna", "atna"),
  iterations = 20L,
  training_fraction = 0.7,
  format = c("auto", "long", "wide"),
  id = "id",
  period = "period",
  state = "state",
  seed = NULL,
  ...
)
```

Arguments

data	Sequence data in long or wide form.
models	Character vector of TNA estimators to compare, any of "tna", "ftna", "ctna", "atna". The default runs all four.
iterations	Number of train/test splits, a whole number of at least 1, default 20.
training_fraction	Fraction of sequences assigned to the training half, strictly between 0 and 1, default 0.7. At least two training sequences are always used, and a split leaving fewer than two testing sequences is an error.
format, id, period, state	Arguments describing the shape and column names of data. They are used to reshape data before splitting; both halves are then fitted in wide form.
seed	Optional single seed, restored on exit.
...	Further arguments passed to <code>fit_tna()</code> .

Value

A plain base data.frame with one row per iteration and estimator and the columns iteration, model, and the `compare_networks()` agreement metrics pearson, cosine, mae, rmse, jaccard, edges_x and edges_y. This is not a `simulab_sim`.

Examples

```
data <- simulate_sequences(n = 60, n_states = 3, chain_length = 12, seed = 1)
if (requireNamespace("tna", quietly = TRUE)) {
  cross_validate_tna(data, models = c("tna", "ftna"), iterations = 2, seed = 1)
}
```

define_condition	<i>Define a conditional data-generation rule</i>
------------------	--

Description

Define a conditional data-generation rule

Usage

```
define_condition(
  variable,
  condition,
  formula,
  variance = 0,
  distribution = .simulab_distributions,
  link = c("identity", "log", "logit")
)
```

Arguments

variable Variable to create or replace.

condition Logical expression identifying affected rows.

formula, variance, distribution, link
 Generation arguments used for affected rows.

Value

A one-row simulab_condition_spec base data.frame.

Examples

```
define_condition(
  "outcome", condition = "group == 1",
  formula = "2", variance = "1", distribution = "normal"
)
```

define_conditions	<i>Combine conditional data-generation rules into a specification</i>
-------------------	---

Description

Combine conditional data-generation rules into a specification

Usage

```
define_conditions(...)
```

Arguments

... One of three forms.

Rules. outcome = when(group == 1, normal(mean = 5, sd = 1)) names the variable with the argument name, states the condition that selects the rows the rule writes, and states their distribution as a call. Any distribution in [list_distributions\(\)](#) may be used, and its parameters may be expressions over the data. Repeating the argument name gives one variable several rules, which is how a variable takes a different distribution in each group. when(TRUE, ...) applies to every row.

Specification columns given as named vectors (variable, condition, formula, variance, distribution, link).

Objects created by [define_condition\(\)](#).

The forms cannot be mixed in one call.

In the column form, variable, condition and formula are required. A column given as a single value is recycled across every rule. variance defaults to 0, distribution to "normal", and link to "identity".

Value

A `simulab_condition_spec` base data.frame. Rules written with `when()` give one row per parameter, with columns `rule`, `variable`, `condition`, `distribution`, `parameter` and `value`; the other two forms give one row per rule with the `formula`/`variance` columns.

Examples

```
# Rules. One line per rule; a repeated name gives a variable several rules.
define_conditions(
  outcome = when(group == 1, normal(mean = 5, sd = 1)),
  outcome = when(group == 0, poisson(lambda = 2)),
  bonus   = when(group == 1, gamma(shape = 2, rate = 0.5))
)

# One call, named arguments, one row per rule.
define_conditions(
  variable = c("outcome", "outcome"),
  condition = c("group == 1", "group == 0"),
  formula   = c("2", "0"),
  variance  = "1"
)

# Definitions built one at a time are still accepted.
define_conditions(
  define_condition("outcome", condition = "group == 1",
                  formula = "2", variance = "1", distribution = "normal"),
  define_condition("outcome", condition = "group == 0",
                  formula = "0", variance = "1", distribution = "normal")
)
```

<code>define_missingness</code>	<i>Define missingness for one variable</i>
---------------------------------	--

Description

Records the rule that decides whether one variable is missing for a row. The rule is a formula evaluated row by row against the data, so missingness may depend on any column: on a fully observed covariate, which is MAR, or on the variable's own value, which is MNAR.

Usage

```
define_missingness(
  variable,
  formula,
  link = c("identity", "logit"),
  baseline = FALSE,
  monotone = FALSE
)
```

Arguments

variable	Variable that may be missing. A single non-empty string.
formula	Missingness formula, given as a number, a string, or an unquoted expression over the data columns. It is read as a probability under the identity link and as log odds under the logit link. It is stored as text and evaluated row by row.
link	Scale the formula is stated on, one of "identity" (the default) or "logit".
baseline	Replace every period's draw for a unit with its first period's draw, so the unit is either always or never missing. A single flag, defaulting to FALSE. It needs id and period in <code>missingness_matrix()</code> .
monotone	Once missing, remain missing at every later period. A single flag, defaulting to FALSE. It needs id and period in <code>missingness_matrix()</code> .

Value

A one-row `simulab_missing_spec` base data.frame with the character columns `variable`, `formula` and `link` and the logical columns `baseline` and `monotone`.

Examples

```
define_missingness("outcome", formula = "0.2")
```

`define_missingnesses` *Combine missingness definitions into a specification*

Description

Combine missingness definitions into a specification

Usage

```
define_missingnesses(...)
```

Arguments

... Either the columns of a specification given as named vectors (`variable`, `formula`, `link`, `baseline`, `monotone`), or objects created by `define_missingness()`. The two forms cannot be mixed in one call.

In the column form, `variable` and `formula` are required. A column given as a single value is recycled across every variable. `link` defaults to "identity", and `baseline` and `monotone` to FALSE.

Value

A `simulab_missing_spec` base data.frame with one row per target variable, in the order given, and the columns of `define_missingness()`: `variable`, `formula`, `link`, `baseline` and `monotone`. Target variables must be unique.

Examples

```
# One call, named arguments, one row per target variable.
define_missingnesses(
  variable = c("outcome", "baseline"),
  formula = c("0.2", "0.1")
)

# `monotone` keeps a unit missing at every later period once it drops out.
define_missingnesses(
  variable = c("outcome", "baseline"),
  formula = c("0.2", "0.1"),
  monotone = TRUE
)

# Definitions built one at a time are still accepted.
define_missingnesses(
  define_missingness("outcome", formula = "0.2"),
  define_missingness("baseline", formula = "0.1")
)
```

define_survival	<i>Define a survival process</i>
-----------------	----------------------------------

Description

Records one Weibull hazard segment for one event time. The `simstudy` parameterization is used throughout: an event time drawn from this segment has survival function $S(t) = \exp(-\exp(\text{formula}) * t^{(1 / \text{shape})} / \text{scale})$, so `shape` is the *reciprocal* of the textbook Weibull shape, and `exp(formula)` multiplies the whole cumulative hazard. A coefficient inside `formula` is therefore a log hazard ratio, while `formula` itself equals the log hazard only when `shape` and `scale` are both 1.

Usage

```
define_survival(event, formula = 0, scale = 1, shape = 1, transition = 0)
```

Arguments

<code>event</code>	Name of the event-time variable. A single non-empty string.
<code>formula</code>	Linear predictor on the log scale, given as a number, a string, or an unquoted expression over the covariate columns (for example <code>"-8 + 0.5 * treatment"</code>). Defaults to 0. It is stored as text and evaluated row by row against the data.
<code>scale</code>	Positive Weibull scale, as a number, a string, or an expression over the covariates. Defaults to 1. It divides the cumulative hazard.
<code>shape</code>	Positive Weibull shape in the <code>simstudy</code> parameterization (the exponent applied to the transformed time, so the hazard is constant when <code>shape</code> is 1 and decreasing when <code>shape</code> is above 1). Defaults to 1.
<code>transition</code>	Time at which this hazard segment begins. A single finite non-negative number, defaulting to 0. The first segment of an event must start at 0.

Value

A one-row `simulab_survival_spec` base data.frame with columns `event`, `formula`, `scale` and `shape` (all character, the stored definition text) and the numeric column `transition`.

Examples

```
define_survival("time", formula = -8, shape = 0.3)
```

<code>define_survivals</code>	<i>Combine survival definitions into a specification</i>
-------------------------------	--

Description

Combine survival definitions into a specification

Usage

```
define_survivals(...)
```

Arguments

... One of three forms.

Hazard calls. `time = hazard(log_rate = -8 + 0.5 * treatment, shape = 0.3)` names the event with the argument name and states its hazard as a call. `log_rate` is the linear predictor on the log scale, the formula of `define_survival()`, so a coefficient inside it is a log hazard ratio. It may be any expression over the covariates; `shape` and `scale` default to 1 and from, the time at which the segment begins, to 0. Arguments may be positional, in the order `log_rate`, `shape`, `scale`, `from`, or named, and repeating the argument name gives one event several segments, which is a piecewise hazard.

Specification columns given as named vectors (`event`, `formula`, `scale`, `shape`, `transition`).

Objects created by `define_survival()`.

The forms cannot be mixed in one call.

In the column form, `event` and `formula` are required. A column given as a single value is recycled across every row. `scale` and `shape` default to 1 and `transition` to 0. Two rows for one event with different transition times give a piecewise hazard.

Value

A `simulab_survival_spec` base data.frame with one row per hazard segment and the columns of `define_survival()`: `event`, `formula`, `scale`, `shape` and `transition`. Rows written by the column form and by the hazard-call form are ordered by event and then transition.

Examples

```
# One call, named arguments, one row per event.
define_survivals(
  event   = c("time_relapse", "time_death"),
  formula = c(-8, -9),
  shape   = 0.3
)

# A piecewise hazard is two rows for one event.
define_survivals(
  event      = c("time", "time"),
  formula    = c(-8, -6),
  shape      = 0.3,
  transition = c(0, 50)
)

# Definitions built one at a time are still accepted.
define_survivals(
  define_survival("time_relapse", formula = -8, shape = 0.3),
  define_survival("time_death", formula = -9, shape = 0.3)
)

# Hazard calls. The log rate is an expression over the covariates.
define_survivals(
  time_relapse = hazard(log_rate = -8 + 0.5 * treatment, shape = 0.3),
  time_death   = hazard(log_rate = -9, shape = 0.3)
)

# A repeated event name is a piecewise hazard: the rate rises after day 60.
define_survivals(
  time = hazard(log_rate = -8, shape = 0.3),
  time = hazard(log_rate = -5, shape = 0.3, from = 60)
)
```

define_variable	<i>Define one simulated variable</i>
-----------------	--------------------------------------

Description

Define one simulated variable

Usage

```
define_variable(
  name,
  formula,
  variance = 0,
  distribution = .simulab_distributions,
  link = c("identity", "log", "logit")
)
```

Arguments

name	Variable name.
formula	Numeric value, expression string, or one-sided formula giving the distribution's primary argument. For most families that is the mean or probability, but "uniform" and "uniform_integer" take "minimum;maximum", "categorical" takes semicolon-separated category probabilities, "mixture" takes "value probability + ...", "treatment" takes semicolon-separated allocation ratios, "cluster_size" takes the total to be split, and "custom" takes the name of a generator function.
variance	Variance, dispersion, precision, trial count, category labels, or distribution-specific secondary parameter.
distribution	Distribution name.
link	Link applied to formula.

Value

A one-row simulab_spec base data.frame.

Examples

```
define_variable(  
  name = "score",  
  formula = 50,  
  variance = 100,  
  distribution = "normal"  
)
```

define_variables	<i>Combine variable definitions into a specification</i>
------------------	--

Description

Combine variable definitions into a specification

Usage

```
define_variables(...)
```

Arguments

... One of three forms.

Distribution calls. `age = normal(mean = 50, sd = 10)` names the variable with the argument name and states its distribution as a call. Parameters may be given positionally or by name, in the order `list_distributions()` reports. A parameter may be any expression over variables defined earlier in the same call, so `outcome = normal(mean = 10 + 0.2 * age, sd = 2)` is a regression. Distribution names are never evaluated, so `gamma()`, `beta()` and `t()` do not reach the base functions of those names.

Specification columns given as named vectors (variable, formula, variance, distribution, link).

Objects created by `define_variable()` or `repeat_variables()`.

The forms cannot be mixed in one call.

In the column form, variable and formula are required. A column given as a single value is recycled across every variable. variance defaults to 0, distribution to "normal", and link to "identity".

Value

A `simulab_spec` base data.frame, shaped by the form it was given. Distribution calls give one row per distribution parameter, with columns variable, distribution, parameter and value. The column form and the constructor form give one row per variable, with columns variable, distribution, formula, variance and link. Both forms are accepted by `simulate_study()` and `augment_study()`.

Examples

```
# Distribution calls. The variable name is the argument name.
define_variables(
  age      = normal(mean = 50, sd = 10),
  treated  = binary(prob = 0.5),
  outcome  = normal(mean = 10 + 0.2 * age + 2 * treated, sd = 2)
)

# Parameters may be positional, in the order list_distributions() reports.
define_variables(y = normal(5, 1), count = poisson(3))

# Specification columns. `formula` is the mean or linear predictor and may
# refer to variables defined earlier. `variance` is a variance, not a
# standard deviation.
define_variables(
  variable   = c("baseline", "treatment", "outcome"),
  formula    = c("0", "0.5", "0.4 * baseline + 0.8 * treatment"),
  variance   = c("1", "0", "1"),
  distribution = c("normal", "binary", "normal")
)

# A column given once is recycled, so a specification that shares one
# distribution stays short.
define_variables(
  variable = c("x1", "x2", "x3"),
  formula  = "0",
  variance = "1"
)

# Definitions built one at a time are still accepted.
define_variables(
  define_variable("age", formula = 40, variance = 100, distribution = "normal"),
  define_variable("treated", formula = 0.5, distribution = "binary")
)
```

encode_factors	<i>Encode categorical variables</i>
----------------	-------------------------------------

Description

Turns categorical columns into the numeric columns a model needs. Dummy coding is one indicator per level, with no reference level dropped; effect coding drops the last level's indicator and codes that level -1 across the remaining columns; factor coding leaves one factor column.

Usage

```
encode_factors(
  data,
  variables,
  coding = c("factor", "dummy", "effect"),
  labels = NULL,
  prefix = "f_",
  replace = FALSE
)
```

Arguments

data	Base data.frame, or a simulab_sim.
variables	Categorical variables to encode. A character vector of at least one column name of data. Levels are taken in the order factor() sorts them.
coding	Encoding to apply, one of "factor" (the default), one factor column named <prefix><variable>, "dummy", one 0/1 indicator per level named <prefix><variable>_<level>, or "effect", the same indicators without the last level's column and with -1 in every column for that level.
labels	Optional replacement level labels, as a named list per variable, one atomic vector shared by every variable, or a tidy data frame with columns variable, level and label. Defaults to NULL, the values as they stand. The labels are passed to factor(), so they rename the levels in sorted order and also feed the generated column names.
prefix	Prefix for generated variables. A single string, defaulting to "f_".
replace	Remove the source variables after encoding. A single flag, defaulting to FALSE.

Value

A simulab_sim base data.frame with the columns of data, minus the source variables when replace is TRUE, followed by the generated columns, one row per input row. The component encoding, reached with as.data.frame(x, what = "encoding"), holds one row per generated column with columns source, encoded and coding.

Examples

```
data <- data.frame(id = 1:6, arm = rep(c("a", "b", "c"), each = 2))
encode_factors(data, variables = "arm", coding = "dummy")
```

encode_sequences	<i>One-hot encode long-form sequence states</i>
------------------	---

Description

One-hot encode long-form sequence states

Usage

```
encode_sequences(
  data,
  id = "id",
  period = "period",
  state = "state",
  prefix = "state_"
)
```

Arguments

- data Long-form sequence data.
- id, period, state Column names.
- prefix Generated state-column prefix.

Value

A `simulab_sim` base `data.frame` with one row per input row: every input column except `state`, followed by one 0/1 indicator column per observed state. A mapping table (`state`, `column`) is available through `as.data.frame()`.

Examples

```
data <- simulate_sequences(n = 10, n_states = 3, chain_length = 5, seed = 1)
head(encode_sequences(data))
```

```
evaluate_edge_recovery
```

Evaluate edge recovery against a known network

Description

Evaluate edge recovery against a known network

Usage

```
evaluate_edge_recovery(truth, estimate, threshold = 0)
```

Arguments

`truth, estimate` Supported network representations.

`threshold` Absolute threshold defining a recovered edge: a dyad counts as present when $\text{abs}(\text{weight}) > \text{threshold}$.

Value

A tidy `simulab_sim` base `data.frame` with one row per dyad in the union of the two edge sets – dyads absent from both networks are not represented, so true negatives are not counted – and columns `from`, `to`, `truth_weight`, `estimate_weight`, `truth_present`, `estimate_present`, `recovered`, `false_positive`, and `false_negative`. A one-row summary table (`precision`, `recall`, `f1`, `true_positive`, `false_positive`, `false_negative`) is available through `as.data.frame()`.

Examples

```
truth <- simulate_network(nodes = 25, model = "bernoulli", probability = 0.15, seed = 1)
estimate <- simulate_network(nodes = 25, model = "bernoulli", probability = 0.15, seed = 2)
evaluate_edge_recovery(truth, estimate)
```

```
evaluate_tna_estimation
```

Evaluate TNA estimation against generated truth

Description

Draws a transition system per replication, simulates sequences from it, fits each estimator in `models`, and scores the fitted edges against the generating probabilities. Fitted weights are row-normalized before the comparison, so estimators on different weight scales are all read as transition probabilities.

Usage

```

evaluate_tna_estimation(
  repetitions = 100L,
  n = 200L,
  chain_length = 25L,
  n_states = 6L,
  models = c("tna", "ftna", "ctna", "atna"),
  concentration = 1,
  diagonal_concentration = 0,
  missing_tail = c(0L, 5L),
  threshold = 0,
  seed = NULL,
  ...
)

```

Arguments

repetitions	Number of simulation replications. A single positive whole number, defaulting to 100.
n	Number of sequences per replication. A single whole number of at least 2, defaulting to 200.
chain_length	Sequence length. A single whole number of at least 2, defaulting to 25.
n_states	Number of states. A single whole number of at least 2, defaulting to 6.
models	TNA estimators to evaluate. A character vector of at least one of "tna", "ftna", "ctna" and "atna", all four by default.
concentration, diagonal_concentration	Transition-system parameters passed to generate_transition_system() , defaulting to 1 and 0.
missing_tail	Trailing missing positions passed to simulate_sequences() , defaulting to c(0L, 5L).
threshold	Absolute weight threshold above which an edge counts as present, in both the agreement and the recovery metrics. A single non-negative number, defaulting to 0.
seed	Optional base seed. A single number, or NULL (the default). Replication <i>i</i> draws its transition system with seed + <i>i</i> - 1 and its sequences with that offset plus 100000.
...	Arguments passed to fit_tna() .

Value

A `simulab_sim` base data.frame with one row per replication and model, holding iteration, model, the [compare_networks\(\)](#) agreement metrics and the [evaluate_edge_recovery\(\)](#) summary. Components: `truth`, the generating probabilities with columns iteration, from, to and weight; and `estimated_edges`, the row-normalized fitted edges with columns iteration, model, from, to and weight.

Examples

```
if (requireNamespace("tna", quietly = TRUE)) {
  evaluate_tna_estimation(
    repetitions = 2, n = 30, chain_length = 10, n_states = 3,
    models = "tna", seed = 1
  )
}
```

expand_clusters	<i>Expand cluster-level rows into unit-level rows</i>
-----------------	---

Description

Repeats every cluster-level row once per unit it contains, so cluster attributes are copied down to the units, and numbers the units.

Usage

```
expand_clusters(data, cluster, size, unit = "id", include_cluster_data = TRUE)
```

Arguments

data	Cluster-level base data.frame, or a simulab_sim, with one row per cluster and at least one row.
cluster	Name of the cluster identifier. A single string naming a column of data.
size	Units per cluster: the name of a cluster-size column of data, or a single positive whole number applied to every cluster.
unit	Name of the new unit identifier, numbered 1 to the total number of units across all clusters rather than restarting within a cluster. A single non-empty string, defaulting to "id".
include_cluster_data	Copy every cluster-level column down to the units. A single flag, defaulting to TRUE. FALSE keeps only the cluster identifier.

Value

A simulab_sim base data.frame with one row per unit, the unit identifier first and then the retained cluster-level columns. The component clusters, reached with `as.data.frame(x, what = "clusters")`, holds one row per input cluster with columns cluster, the identifier values under that fixed name whatever cluster was called, and the integer size.

Examples

```
clusters <- data.frame(cluster = 1:5, site = rep(c("north", "south"), length.out = 5))
result <- expand_clusters(clusters, cluster = "cluster", size = 4)
head(result)
```

expand_periods	<i>Expand observations across periods</i>
----------------	---

Description

Repeats every unit's row once per period, numbering the periods and adding the elapsed time each period falls at. Units may be followed for different numbers of periods, and the spacing may be regular or irregular. A set of wide columns, one per period, can be gathered into a single long value column at the same time.

Usage

```
expand_periods(
  data,
  periods = NULL,
  id = "id",
  period = "period",
  period_values = NULL,
  interval_mean = NULL,
  interval_dispersion = 0,
  time = "time",
  time_variables = NULL,
  value = "value",
  seed = NULL
)
```

Arguments

data	Base data.frame, or a simulab_sim, with one row per observational unit and at least one row.
periods	Number of periods: a single whole number applied to every unit, the name of a column of data holding a count per unit, or NULL when period_values is supplied.
id	Identifier variable. A single string naming a column of data. It is carried into the schedule component; the returned rows are the input rows repeated, so the identifier is not renumbered.
period	Name of the period column. A single non-empty string, defaulting to "period". Periods generated from periods are numbered from 0, so a unit followed for k periods gets 0 to k - 1.
period_values	Optional atomic vector of period values shared by every unit, used in place of the numbering above. Defaults to NULL, and may not be combined with periods.
interval_mean	Mean interval between consecutive periods: a single number, the name of a column of data, or NULL (the default) for an interval of 1.

interval_dispersion	Dispersion of the gamma-distributed intervals: a single non-negative number, or the name of a column of data. Defaults to 0, exactly regular spacing; a positive value draws each interval from a gamma distribution with mean interval_mean and variance $\text{interval_mean}^2 * \text{interval_dispersion}$.
time	Name of the generated elapsed-time column, which starts at 0 for every unit and accumulates the intervals. A single string, defaulting to "time".
time_variables	Optional character vector of wide columns of data, exactly one per period, gathered into the long value column and then dropped. Defaults to NULL.
value	Name of the gathered time-varying value. A single non-empty string, defaulting to "value".
seed	Optional random seed for irregular intervals. A single number, or NULL (the default).

Value

A `simulab_sim` base `data.frame` with one row per unit-period, holding the columns of data plus the period and elapsed-time columns, and the gathered value column in place of `time_variables` when those are given. The component `schedule`, reached with `as.data.frame(x, what = "schedule")`, holds one row per input unit with columns `input_row`, `id`, `periods`, `interval_mean` and `interval_dispersion`.

Examples

```
data <- data.frame(id = 1:10)
result <- expand_periods(data, periods = 4, id = "id", period = "period")
head(result)
```

factorial_design	<i>Create a full factorial design</i>
------------------	---------------------------------------

Description

Crosses every level of every factor, repeats each combination replications times, and returns one numeric column per factor holding its coded level.

Usage

```
factorial_design(
  factors,
  replications = 1L,
  coding = c("dummy", "effect", "level"),
  id = "id"
)
```

Arguments

factors	Named numeric vector of whole numbers giving the number of levels per factor, each at least 2. The names become the design columns.
replications	Number of replications per factor combination. A single positive whole number, defaulting to 1.
coding	Numeric coding of the levels, one value per factor per row, one of "dummy" (the default), which codes the levels 0 to levels - 1, "effect", which codes a two-level factor -1 and 1 and codes a factor with more levels exactly as "dummy" does, or "level", which keeps the level numbers 1 to levels.
id	Identifier-column name, numbering the rows. A single non-empty string, defaulting to "id".

Value

A `simulab_sim` base data.frame with one row per replicated factor combination (`replications * prod(factors)` rows), the identifier column first and then one coded column per factor. The component factors, reached with `as.data.frame(x, what = "factors")`, holds one row per factor with columns factor, the integer levels and coding.

Examples

```
factorial_design(factors = c(dose = 3L, timing = 2L), replications = 2)
```

fit_tna

Fit a temporal network analysis model

Description

Fit a temporal network analysis model

Usage

```
fit_tna(
  data,
  model = c("tna", "ftna", "ctna", "atna"),
  format = c("auto", "long", "wide"),
  id = "id",
  period = "period",
  state = "state",
  group = NULL,
  ...
)
```

Arguments

data	Long- or wide-form sequence data.
model	TNA, frequency TNA, co-occurrence TNA, or attention TNA.
format	Input format. auto recognizes canonical long data.
id, period, state	Canonical long-form column names.
group	Optional grouping-column name. Supplying it fits a model per group through the corresponding grouped TNA estimator.
...	Additional arguments passed to the estimator in tna.

Value

A tidy edge-list simulab_sim base data.frame with one row per ordered state pair and columns from, to, and weight, prefixed by a group column when group is supplied. For model = "tna" the weights are the estimated transition probabilities and sum to one within each from state; the other estimators return their own (not row-stochastic) weights. as.data.frame(x, what =) also returns initial_probabilities (state, probability) and model_info (model, grouped, group, observations, sequence_positions). Use as_tna_model() when a native model object is required by tna plotting or inference functions.

Examples

```
data <- simulate_sequences(n = 40, n_states = 3, chain_length = 12, seed = 1)
if (requireNamespace("tna", quietly = TRUE)) {
  head(fit_tna(data, model = "tna"))
}
```

fit_tna_batch

*Fit TNA models to multiple datasets***Description**

Fit TNA models to multiple datasets

Usage

```
fit_tna_batch(inputs, ...)
```

Arguments

inputs	List of sequence data frames, with at least one element. Element names label the datasets; when inputs is unnamed the labels "Dataset 1", "Dataset 2", ... are used instead.
...	Arguments passed to fit_tna() .

Value

A `simulab_sim` edge list with one row per dataset and transition: a leading dataset column followed by the columns `fit_tna()` returns (from, to, weight, and group when a grouped fit is requested). The `model_info` component holds one row per dataset describing the fitted model.

Examples

```
inputs <- list(
  first = simulate_sequences(n = 30, n_states = 3, chain_length = 10, seed = 1),
  second = simulate_sequences(n = 30, n_states = 3, chain_length = 10, seed = 2)
)
if (requireNamespace("tna", quietly = TRUE)) {
  head(fit_tna_batch(inputs, model = "tna"))
}
```

```
generate_transition_system
```

Generate a transition system

Description

Generate a transition system

Usage

```
generate_transition_system(
  n_states = 8L,
  states = NULL,
  concentration = 1,
  diagonal_concentration = 0,
  state_categories = c("metacognitive", "cognitive"),
  seed = NULL,
  batch = NULL
)
```

Arguments

<code>n_states</code>	Number of states.
<code>states</code>	Optional state labels.
<code>concentration</code>	Dirichlet concentration applied to every transition and to the initial-state probabilities.
<code>diagonal_concentration</code>	Optional extra concentration added to the self-transition (diagonal) entries only. The default 0 leaves self-transitions no more likely than any other transition.

state_categories	Learning-state categories the state labels are drawn from with <code>sample_learning_states()</code> when <code>states</code> is <code>NULL</code> , defaulting to <code>c("metacognitive", "cognitive")</code> ; see learning_state_categories() . <code>NULL</code> labels the states <code>State 1</code> , <code>State 2</code> , and so on.
seed	Optional random seed.
batch	Optional single positive whole number. When given, the simulator runs batch times and returns a plain list of batch results, each exactly what the same call without batch returns. With a seed, every dataset gets its own seed drawn from seed, so the whole batch is reproducible; without one the datasets are consecutive draws from the session's random-number stream. The default <code>NULL</code> returns a single result.

Value

A tidy transition-edge `simulab_sim` base `data.frame` with n_states^2 rows and columns `from`, `to`, and `probability`; every `from` state's probabilities sum to one. An `initial_probabilities` table (`state`, `probability`) is available through `as.data.frame()`.

Examples

```
result <- generate_transition_system(n_states = 3, seed = 1)
result
```

global_name_regions	<i>List global-name regions</i>
---------------------	---------------------------------

Description

List global-name regions

Usage

```
global_name_regions()
```

Value

A character vector of available region identifiers.

Examples

```
global_name_regions()
```

global_names	List the global-name catalogue
--------------	--------------------------------

Description

List the global-name catalogue

Usage

```
global_names(regions = "all")
```

Arguments

regions	Region names from <code>global_name_regions()</code> , or "all", the default, for every region.
---------	---

Value

A tidy base data.frame with one row per region/name pair and columns region and name. A name shared by two regions appears once per region.

Examples

```
head(global_names())  
head(global_names(regions = c("east_asia", "africa")))
```

inject_missingness	Inject MCAR, MAR, or MNAR missingness
--------------------	---------------------------------------

Description

Blanks cells of data under one of the three standard mechanisms. "MCAR" gives every row the same probability proportion, so missingness depends on nothing. "MAR" makes it depend on the fully observed predictor, which is itself never made missing. "MNAR" makes each target variable's missingness depend on its own value, so the values that disappear are not a random sample of them. For MAR and MNAR the probability increases with the driving variable's rank: its ties-averaged rank mapped into (0, 1), or 0.5 for a missing or constant driver, scaled by the multiplier that makes the mean probability equal proportion, and capped at 1. Each MAR target therefore shares one probability vector, while each MNAR target gets its own.

Usage

```
inject_missingness(
  data,
  mechanism = c("MCAR", "MAR", "MNAR"),
  proportion = 0.1,
  variables = NULL,
  predictor = NULL,
  seed = NULL
)
```

Arguments

<code>data</code>	Complete base <code>data.frame</code> , or a <code>simulab_sim</code> , with at least one row.
<code>mechanism</code>	Missingness mechanism, one of "MCAR" (the default), "MAR" or "MNAR".
<code>proportion</code>	Target missing fraction, the mean cell probability rather than a realized count. A single number between 0 and 1, defaulting to 0.1; the realized fraction is random around it and is reported in the <code>missingness_summary</code> component.
<code>variables</code>	Variables to make missing. A character vector of column names, or NULL (the default), which selects every column of data, identifiers included. Under "MAR" the predictor is removed from this set.
<code>predictor</code>	Fully observed predictor that drives MAR missingness. A single string naming a column of data, required when mechanism is "MAR" and ignored otherwise. Defaults to NULL.
<code>seed</code>	Optional random seed. A single number, or NULL (the default).

Value

A `simulab_sim` base `data.frame` with the columns and rows of data, the drawn cells of the target variables replaced by NA. Components: `missingness`, the cell-level mask reached with `as.data.frame(x, what = "missingness")`, one row per observation and target variable with columns `observation`, `variable`, `probability`, the logical missing and mechanism; and `missingness_summary`, one row per target variable with columns `variable`, `realized_proportion` and `target_proportion`.

Examples

```
data <- data.frame(id = 1:200, x = stats::rnorm(200), y = stats::rnorm(200))

result <- inject_missingness(
  data, mechanism = "MCAR", proportion = 0.2, variables = "y", seed = 1
)
head(result)

# MAR missingness in `y` driven by the observed predictor `x`.
head(inject_missingness(
  data, mechanism = "MAR", proportion = 0.2,
  variables = "y", predictor = "x", seed = 1
))
```

```
learning_state_categories
```

List learning-state categories

Description

List learning-state categories

Usage

```
learning_state_categories()
```

Value

A character vector of available category identifiers.

Examples

```
learning_state_categories()
```

```
learning_states
```

List categorized learning states for sequence simulation

Description

List categorized learning states for sequence simulation

Usage

```
learning_states(categories = "all")
```

Arguments

categories A character vector of one or more category identifiers from [learning_state_categories\(\)](#), or "all" (the default) for every category. Unknown identifiers raise an error.

Value

A tidy base `data.frame` with one row per category-state pair and columns `category` and `state`. A state may belong to more than one category and then appears once per category, so `state` is not unique: the full catalogue has 209 rows covering 202 distinct states.

Examples

```
head(learning_states())
head(learning_states(categories = "metacognitive"))
```

linear_formula	<i>Construct a linear simulation formula</i>
----------------	--

Description

Construct a linear simulation formula

Usage

```
linear_formula(coefficients, variables)
```

Arguments

coefficients	Coefficients with an optional intercept.
variables	Variable names. When there is one additional coefficient, the first is the intercept.

Value

A one-row simulab_formula base data.frame.

Examples

```
linear_formula(coefficients = c(1, 0.5, -0.3), variables = c("x1", "x2"))
```

list_distributions	<i>List the distributions a specification may use</i>
--------------------	---

Description

list_distributions() reports the distribution catalogue. Each row names one distribution, the parameters it takes in the order a positional call supplies them, and whether it carries a quantile function. Only a distribution with a quantile function can be a marginal in [simulate_copula\(\)](#).

Usage

```
list_distributions(pattern = NULL, copula = NULL)
```

Arguments

pattern	Optional regular expression matched against distribution names.
copula	If TRUE, keep only distributions usable as copula marginals; if FALSE, keep only those that are not. NULL, the default, keeps every distribution.

Value

A base data.frame with one row per distribution and columns distribution, parameters, n_parameters and copula.

Examples

```
head(list_distributions())

# Distributions whose name mentions normal.
list_distributions("normal")

# Distributions that cannot be a copula marginal.
list_distributions(copula = FALSE)
```

list_simulators	<i>List public simulation and generation verbs</i>
-----------------	--

Description

List public simulation and generation verbs

Usage

```
list_simulators(family = NULL, kind = NULL, dispatchable = NULL)
```

Arguments

family	Optional single-string family filter, one of "general", "statistical", "latent", "longitudinal", "measurement", "sequence", "survival", "empirical", "functional" or "network". An unknown family is an error.
kind	Optional single-string kind filter: "simulator", "generator", "workflow", or "dispatcher". An unknown kind is an error.
dispatchable	Optional single logical filter: TRUE keeps only the verbs that can be called through simulate_data() , FALSE keeps only those that cannot.

Value

A base data.frame with one row per catalogued verb and the columns simulator (the catalogue name), function_name, kind, family, primary_shape and dispatchable.

Examples

```
head(list_simulators())
list_simulators(kind = "workflow")
```

merge_studies	<i>Merge study data by identifiers</i>
---------------	--

Description

Merge study data by identifiers

Usage

```
merge_studies(x, y, by, join = c("inner", "full", "left"))
```

Arguments

x, y	Base data frames, or simulab_sim objects, to join.
by	Identifier variables shared by both inputs. A character vector of at least one name present in both.
join	Join type, one of "inner" (the default), keeping only matched identifiers, "full", keeping every row of both inputs, or "left", keeping every row of x.

Value

A simulab_sim base data.frame containing the merged data, sorted by by as merge() leaves it, with unmatched cells filled with NA under the "full" and "left" joins.

Examples

```
x <- data.frame(id = 1:5, baseline = 1:5)
y <- data.frame(id = 1:5, outcome = 6:10)
merge_studies(x, y, by = "id")
```

missingness_matrix	<i>Generate a missingness mask</i>
--------------------	------------------------------------

Description

Draws, for every row of data and every variable in specification, whether that cell is missing, by comparing a uniform draw against the row's probability. The mask is returned rather than applied, so the complete data and the mask that hides part of it can both be kept; `observed_data()` joins them.

Usage

```
missingness_matrix(
  data,
  specification,
  id = NULL,
  period = NULL,
  seed = NULL,
  envir = parent.frame()
)
```

Arguments

<code>data</code>	Complete base <code>data.frame</code> , or a <code>simulab_sim</code> , with at least one row. Every target variable of specification must be one of its columns.
<code>specification</code>	Definitions from <code>define_missingnesses()</code> , a <code>simulab_missing_spec</code> object.
<code>id</code>	Optional unit identifier. A single string naming a column of data, or <code>NULL</code> (the default). It is required by the baseline and monotone rules and is carried into the mask.
<code>period</code>	Optional period variable that orders a unit's rows. A single string naming a column of data, or <code>NULL</code> (the default). It is required by the baseline and monotone rules and is carried into the mask.
<code>seed</code>	Optional random seed. A single number, or <code>NULL</code> (the default).
<code>envir</code>	Environment the formulas are evaluated in after the data columns. Defaults to the caller's environment.

Value

A base `data.frame` with one row per row of data, holding the `id` and `period` columns when they are given, or a single row column numbering the rows when neither is, followed by one logical column per target variable of specification, named after that variable. Variables of data that specification says nothing about get no column.

Examples

```
data <- data.frame(id = 1:100, baseline = stats::rnorm(100), outcome = stats::rnorm(100))
specification <- define_missingnesses(
  define_missingness("outcome", formula = "0.2")
)

mask <- missingness_matrix(data, specification = specification, seed = 1)
head(mask)
```

mixture_formula	<i>Construct a mixture simulation formula</i>
-----------------	---

Description

Construct a mixture simulation formula

Usage

```
mixture_formula(variables, probabilities = NULL)
```

Arguments

`variables` Variable names or expressions.
`probabilities` Optional probabilities. Equal probabilities are used by default.

Value

A one-row `simulab_formula` base `data.frame`.

Examples

```
mixture_formula(variables = c("a", "b"), probabilities = c(0.3, 0.7))
```

network_centrality	<i>Calculate tidy network centralities</i>
--------------------	--

Description

Calculate tidy network centralities

Usage

```
network_centrality(
  network,
  measures = c("degree", "strength", "betweenness", "closeness", "eigenvector",
    "pagerank"),
  directed = NULL
)
```

Arguments

network	A supported network representation.
measures	Centrality measures.
directed	Treat edges as directed. NULL (the default) takes it from the network's settings table, as <code>as_igraph()</code> does. This governs the graph conversion and the betweenness, eigenvector, and pagerank measures only; degree, strength, and closeness always combine incoming and outgoing ties. betweenness and closeness use $1 / \text{weight} $ as the edge distance.

Value

A tidy base data.frame with one row per node and measure and columns node, measure, and value, stacked measure after measure. Requires the suggested igraph package.

Examples

```
network <- simulate_network(nodes = 30, model = "bernoulli", probability = 0.1, seed = 1)
if (requireNamespace("igraph", quietly = TRUE)) {
  head(network_centrality(network, measures = c("degree", "strength")))
}
```

observe_treatment	<i>Generate observational treatment or exposure groups</i>
-------------------	--

Description

Evaluates one formula per modeled group against every row of data and draws that row's exposure from the resulting probabilities, so the formulas are the propensity model. One group is always implicit: it is listed last and takes the remaining probability, which makes it the reference category of the logit link.

Usage

```
observe_treatment(
  data,
  formulas,
  link = c("identity", "logit"),
  labels = NULL,
  name = "treatment",
  seed = NULL,
  envir = parent.frame()
)
```

Arguments

data	Base data.frame, or a simulab_sim, with at least one row.
formulas	Character vector of at least one formula, each an expression over the columns of data. Under the identity link they are evaluated on the probability scale; under the logit link they are the log odds of that group relative to the implicit final group, that is a multinomial logit linear predictor. An implicit final group receives the remaining probability.
link	Scale the formulas are stated on, one of "identity" (the default) or "logit", the multinomial-logit link.
labels	Optional atomic vector of group labels, one per formula plus one for the implicit final group. Defaults to NULL: c(1, 0) for a single formula, so the modeled group is 1 and the implicit reference is 0, and 1 to the number of groups otherwise, the implicit group last.
name	Name of the exposure variable, which must not already exist in data. A single non-empty string, defaulting to "treatment".
seed	Optional random seed. A single number, or NULL (the default).
envir	Environment the formulas are evaluated in after the data columns. Defaults to the caller's environment.

Value

A simulab_sim base data.frame with the columns of data followed by the exposure variable named by name, one row per input row. The component probabilities, reached with `as.data.frame(x, what = "probabilities")`, holds one row per observation and group, with columns observation, group and probability.

Examples

```
data <- data.frame(id = 1:100, severity = stats::rnorm(100))

result <- observe_treatment(
  data, formulas = "-0.5 + 0.8 * severity", link = "logit", seed = 1
)
head(result)
```

observed_data	<i>Apply a missingness mask to complete data</i>
---------------	--

Description

Sets every cell the mask flags to NA, leaving the complete data otherwise untouched, and keeps the mask alongside the result as a tidy component.

Usage

```
observed_data(data, missingness, id = NULL)
```

Arguments

data	Complete base data.frame, or a simulab_sim, with as many rows as missingness and in the same row order.
missingness	A logical mask returned by <code>missingness_matrix()</code> , with one row per row of data. Its columns that also name columns of data, other than those listed in id, are the targets, and each must be logical.
id	Identifier columns the mask carries that are never made missing. A character vector, or NULL (the default). Name the identifier columns of the mask here, or they are treated as targets and rejected for not being logical.

Value

A simulab_sim base data.frame with the columns and rows of data, the flagged cells of the target variables replaced by NA. The component missingness, reached with `as.data.frame(x, what = "missingness")`, is the mask in long form, one row per observation and target variable, with columns observation, variable and the logical missing.

Examples

```
data <- data.frame(id = 1:100, outcome = stats::rnorm(100))
mask <- missingness_matrix(
  data,
  specification = define_missingnesses(define_missingness("outcome", formula = "0.3")),
  seed = 1
)

result <- observed_data(data, missingness = mask, id = "id")
head(result)
```

parameter_grid

Generate parameter combinations

Description

Generate parameter combinations

Usage

```
parameter_grid(
  ...,
  n = 10L,
  method = c("grid", "random", "latin_hypercube"),
  seed = NULL
)
```

Arguments

...	Named vectors, one per parameter. Under method = "grid" every element is a level of a full factorial crossing. Under the sampling methods a length-2 numeric vector is read as a c(minimum, maximum) range to draw from, and any other vector is sampled from with replacement.
n	Single positive whole number of rows to draw for the "random" and "latin_hypercube" methods, default 10. Ignored by "grid", whose row count is the product of the parameter lengths.
method	One of "grid" (the default), "random", or "latin_hypercube".
seed	Optional single random seed. It is applied for the sampling methods only and is restored on exit.

Value

A base data.frame with one row per parameter combination, a leading scenario_id column numbering the rows, and then one column per parameter in the order given.

Examples

```
# Full factorial grid over named parameters.
parameter_grid(sample_size = c(50, 100), effect = c(0.2, 0.5))

# Latin-hypercube sample of `n` draws over two ranges. Grid parameters are
# passed through `...`, so none of them may be called `n`, `method` or `seed`.
parameter_grid(
  sample_size = c(50, 200), effect = c(0, 1),
  n = 5, method = "latin_hypercube", seed = 1
)
```

print.simulab_sim	<i>Print a simulation result</i>
-------------------	----------------------------------

Description

Prints a one-line header giving the simulation type and the full dimensions, then at most the first 10 rows of the primary data, then a count of the rows not shown. The table of generating values follows under the heading Truth: the first of coefficients, fixed_effects, parameters, transition, transitions, true_transitions, definitions, probabilities, survival_definitions and settings that the result stores, again at most 10 rows. A last line names the other stored tables, which as.data.frame() returns with what.

Usage

```
## S3 method for class 'simulab_sim'
print(x, ...)
```

Arguments

`x` A simulab_sim object.

`...` Arguments passed to the base data-frame print method.

Value

`x`, invisibly.

Examples

```
result <- simulate_ttest(n_a = 30, n_b = 30, mean_a = 0, mean_b = 0.5, seed = 1)
print(result)
```

print.simulab_tna	<i>Print a simulated transition network</i>
-------------------	---

Description

Prints the fitted network with the native tna print method, then the transition matrix and initial probabilities that generated the sequences, in the same format, then the names of the other stored tables.

Usage

```
## S3 method for class 'simulab_tna'
print(x, digits = 3L, ...)
```

Arguments

`x` A simulab_tna result of `simulate_tna()`.

`digits` Digits for the generating probabilities, default 3.

`...` Passed to the tna print method.

Value

`x`, invisibly.

Examples

```
if (requireNamespace("tna", quietly = TRUE)) {
  print(simulate_tna(n = 30, n_states = 3, seed = 1))
}
```

read_definitions	<i>Read variable definitions from a CSV file</i>
------------------	--

Description

Read variable definitions from a CSV file

Usage

```
read_definitions(file)
```

Arguments

file	Path to a CSV file containing variable, distribution, formula, variance, and link columns.
------	--

Value

A simulab_spec base data.frame.

Examples

```
file <- tempfile(fileext = ".csv")
utils::write.csv(
  data.frame(
    variable = c("baseline", "outcome"),
    distribution = c("normal", "normal"),
    formula = c("0", "0.5 * baseline"),
    variance = c("1", "1"),
    link = c("identity", "identity")
  ),
  file, row.names = FALSE
)

specification <- read_definitions(file)
specification
head(simulate_study(n = 50, specification = specification, seed = 1))
unlink(file)
```

repeat_variables	<i>Define repeated variables</i>
------------------	----------------------------------

Description

Define repeated variables

Usage

```
repeat_variables(
  n,
  prefix,
  formula,
  variance = 0,
  distribution = .simulab_distributions,
  link = c("identity", "log", "logit")
)
```

Arguments

`n` Number of variables.

`prefix` Variable-name prefix.

`formula, variance, distribution, link`
 Arguments passed to `define_variable()`.

Value

A `simulab_spec` base data.frame with one row per repeated variable.

Examples

```
specification <- repeat_variables(
  n = 3, prefix = "item", formula = "0", variance = "1", distribution = "normal"
)
specification
head(simulate_study(n = 20, specification = specification, seed = 1))
```

```
run_simulation_scenario
```

Run a built-in simulation scenario

Description

Run a built-in simulation scenario

Usage

```
run_simulation_scenario(scenario, seed = NULL)
```

Arguments

`scenario` A single scenario name from `simulation_scenarios()`.

`seed` Optional seed.

Value

The `simulab_sim` base `data.frame` returned by the simulator the scenario calls, whose columns depend on the scenario's family.

Examples

```
result <- run_simulation_scenario("small_effect", seed = 1)
head(result)
```

sample_global_names	<i>Sample globally diverse names</i>
---------------------	--------------------------------------

Description

Sample globally diverse names

Usage

```
sample_global_names(n, regions = "all", seed = NULL)
```

Arguments

<code>n</code>	Number of unique names to draw, at most the number of distinct names the chosen regions hold.
<code>regions</code>	Regions passed to <code>global_names()</code> .
<code>seed</code>	Optional seed.

Value

A tidy base `data.frame` with one row per sampled name and columns `order`, `region` and `name`. A name belonging to more than one region carries them joined by `";"`.

Examples

```
sample_global_names(n = 8, seed = 1)
```

`sample_learning_states`*Sample learning states reproducibly*

Description

Samples without replacement from the distinct states of the selected categories, so the result never repeats a state.

Usage

```
sample_learning_states(n, categories = "all", seed = NULL)
```

Arguments

<code>n</code>	Number of unique states to draw. A single positive whole number, at most the number of distinct states in the selected categories.
<code>categories</code>	Categories passed to <code>learning_states()</code> , defaulting to "all".
<code>seed</code>	Optional random seed.

Value

A tidy base data.frame with `n` rows, one per sampled state, and columns `order` (the selection order, 1:n), `category` and `state`. A state belonging to several categories gets all of them in `category`, joined by ";".

Examples

```
sample_learning_states(n = 6, categories = "cognitive", seed = 1)
```

`sample_tna`*Fit TNA to a sequence sample*

Description

Fit TNA to a sequence sample

Usage

```
sample_tna(
  data,
  fraction = 0.3,
  replace = FALSE,
  seed = NULL,
  format = c("auto", "long", "wide"),
  id = "id",
  period = "period",
  state = "state",
  group = NULL,
  ...
)
```

Arguments

<code>data</code>	Sequence data, in long or wide form.
<code>fraction</code>	Single number in $(0, 1]$ giving the fraction of sequences to draw, default 0.3. At least two sequences are always drawn.
<code>replace</code>	Single flag, whether to draw with replacement. Default FALSE.
<code>seed</code>	Optional single seed, restored on exit.
<code>format</code>	One of "auto", "long" or "wide", naming the layout of data. The default "auto" detects it.
<code>id, period, state</code>	Column names identifying the sequence, the position within it, and the state. Used only when data is in long form; the defaults are "id", "period" and "state".
<code>group</code>	Optional column name. When given, sequences are sampled within each group and the refit is grouped.
<code>...</code>	Further arguments passed to <code>fit_tna()</code> , such as <code>model</code> . The sampled sequences are always refitted in wide form, so <code>format</code> is consumed by the sampling step and is not forwarded.

Value

The `fit_tna()` result for the sampled sequences: a `simulab_sim` edge list with one row per transition and the columns `from`, `to` and `weight`, plus the `initial_probabilities` and `model_info` components.

Examples

```
data <- simulate_sequences(n = 40, n_states = 3, chain_length = 12, seed = 1)
if (requireNamespace("tna", quietly = TRUE)) {
  head(sample_tna(data, fraction = 0.5, seed = 1))
}
```

scenario_grid	<i>Build a replicated scenario grid</i>
---------------	---

Description

Crosses every combination of the supplied scenario values and repeats each combination replications times, giving the grid a simulation study is run over.

Usage

```
scenario_grid(..., replications = 1L, id = "scenario_id")
```

Arguments

...	Named vectors of scenario values, at least one, crossed with <code>expand.grid()</code> . The names become the value columns.
replications	Replications per scenario combination. A single positive whole number, defaulting to 1.
id	Name of the scenario identifier, numbering the distinct combinations. A single non-empty string, defaulting to "scenario_id".

Value

A plain base `data.frame`, not a `simulab_sim`, with one row per scenario replication and columns: the scenario identifier named by `id`, the integer replication counting from 1 within each scenario, and one column per named argument.

Examples

```
scenario_grid(mean_b = c(0, 0.5), n_b = c(20, 40), replications = 2)
```

select_variables	<i>Select or remove study variables</i>
------------------	---

Description

Select or remove study variables

Usage

```
select_variables(data, keep = NULL, drop = NULL)
```

Arguments

data	Base data.frame, or a simulab_sim.
keep	Variables to retain, in the order given. A character vector of column names, or NULL (the default).
drop	Variables to remove, the others keeping their order. A character vector of column names, or NULL (the default). Only one of keep and drop may be given.

Value

A simulab_sim base data.frame with every row of data and the requested variables.

Examples

```
data <- data.frame(id = 1:5, keep_me = 1:5, drop_me = 6:10)
select_variables(data, keep = c("id", "keep_me"))
select_variables(data, drop = "drop_me")
```

simulate_anova	<i>Simulate a one-way group design</i>
----------------	--

Description

Simulate a one-way group design

Usage

```
simulate_anova(
  n = 30L,
  means = NULL,
  sds = 1,
  labels = NULL,
  outcome = "outcome",
  seed = NULL,
  batch = NULL
)
```

Arguments

n	Group sample size or one size per group, defaulting to 30.
means	Group means. NULL (the default) draws one mean per group from the standard normal distribution, rounded to two decimals, inside the same seeded draw as the data. The number of groups is then the length of n, sds or labels, or 3 when all of them have length one.
sds	Group standard deviations.
labels	Optional group labels.

outcome	Name of the outcome variable.
seed	Optional random seed.
batch	Optional single positive whole number. When given, the simulator runs batch times and returns a plain list of batch results, each exactly what the same call without batch returns. With a seed, every dataset gets its own seed drawn from seed, so the whole batch is reproducible; without one the datasets are consecutive draws from the session's random-number stream. The default NULL returns a single result.

Value

A `simulab_sim` base data.frame with one row per observation and columns `id`, `group` and `outcome` (renamed by `outcome`). Two tidy tables come from `as.data.frame()`: `what = "parameters"` has one row per group with columns `group`, `n`, `mean` and `sd`; `what = "effects"` has a single row with columns `grand_mean`, `between_sum_squares` ($\sum(n * (\text{mean} - \text{grand_mean})^2)$), `within_sum_squares` ($\sum((n - 1) * \text{sd}^2)$) and `eta_squared`, their ratio `between / (between + within)`. Because the within term uses the degrees of freedom `n - 1` rather than `n`, `eta_squared` is the design's sum-of-squares `eta-squared` and differs slightly from the variance-ratio population `eta-squared`.

Examples

```
result <- simulate_anova(n = 90, means = c(0, 0.4, 0.9), seed = 1)
head(result)
as.data.frame(result, what = "parameters")
```

```
simulate_bipartite_network
      Simulate a bipartite network
```

Description

Simulate a bipartite network

Usage

```
simulate_bipartite_network(
  actors,
  events,
  edges = NULL,
  probability = 0.1,
  weight = c("binary", "uniform", "normal", "poisson"),
  weight_mean = 1,
  weight_sd = 1,
  weight_range = c(0.1, 1),
  seed = NULL,
  batch = NULL
)
```

Arguments

actors	Number or labels of primary-mode nodes.
events	Number or labels of secondary-mode nodes.
edges	Optional exact number of cross-mode edges.
probability	Cross-mode edge probability when edges is NULL.
weight	Edge-weight distribution.
weight_mean, weight_sd, weight_range	Weight parameters.
seed	Optional seed.
batch	Optional single positive whole number. When given, the simulator runs batch times and returns a plain list of batch results, each exactly what the same call without batch returns. With a seed, every dataset gets its own seed drawn from seed, so the whole batch is reproducible; without one the datasets are consecutive draws from the session's random-number stream. The default NULL returns a single result.

Value

A tidy cross-mode edge-list `simulab_sim` base `data.frame` with one row per actor-event edge and columns from (always an actor), to (always an event), weight, from_mode, and to_mode. The two modes are disjoint, so no edge joins two actors or two events and self-loops cannot occur. `as.data.frame(x, what = )` also returns nodes (node, mode), incidence (the wide actor-by-event incidence table: an actor column followed by one column per event), and a one-row settings table.

Examples

```
result <- simulate_bipartite_network(actors = 20, events = 6, probability = 0.3, seed = 1)
head(result)
components(result)
```

simulate_clusters	<i>Simulate Gaussian clusters</i>
-------------------	-----------------------------------

Description

Simulate Gaussian clusters

Usage

```
simulate_clusters(
  n,
  centers,
  sds = 1,
  proportions = NULL,
```

```

    labels = NULL,
    seed = NULL,
    batch = NULL
  )

```

Arguments

n	Sample size.
centers	Matrix with one row per cluster and one column per variable, or a tidy data frame with columns <code>cluster</code> , <code>variable</code> and <code>center</code> .
sds	Scalar, per-variable vector, cluster-by-variable matrix, or a tidy data frame with columns <code>cluster</code> , <code>variable</code> and <code>sd</code> .
proportions	Optional cluster proportions.
labels	Optional cluster labels.
seed	Optional random seed.
batch	Optional single positive whole number. When given, the simulator runs batch times and returns a plain list of batch results, each exactly what the same call without batch returns. With a seed, every dataset gets its own seed drawn from seed, so the whole batch is reproducible; without one the datasets are consecutive draws from the session's random-number stream. The default NULL returns a single result.

Value

A `simulab_sim` base `data.frame` with one row per observation and columns `id`, `cluster` (the true membership label) and one column per variable. One tidy table comes from `as.data.frame()`: `what = "parameters"` has one row per cluster-variable combination with columns `cluster`, `variable`, `center`, `sd` and `proportion`.

Examples

```

# `centers` is one row per cluster and one column per variable.
result <- simulate_clusters(
  n = 150,
  centers = matrix(c(0, 0, 4, 4, 0, 4), nrow = 3, byrow = TRUE),
  sds = 1,
  seed = 1
)
head(result)
as.data.frame(result, what = "parameters")

```

simulate_copula	<i>Simulate correlated general-distribution variables with a Gaussian copula</i>
-----------------	--

Description

Simulate correlated general-distribution variables with a Gaussian copula

Usage

```
simulate_copula(
  n,
  specification,
  rho = 0,
  tau = NULL,
  structure = c("independent", "exchangeable", "ar1", "custom"),
  correlation = NULL,
  id = "id",
  seed = NULL,
  envir = parent.frame(),
  batch = NULL
)
```

Arguments

n	Number of observations.
specification	Variable definitions from <code>define_variables()</code> . In the distribution-call form every marginal must carry a quantile function; <code>list_distributions(copula = TRUE)</code> reports which do. The formula/variance column form covers twelve marginals: formula is the mean (through link) and variance carries the second parameter, which is a variance for normal, a precision for beta, a dispersion for gamma and negative_binomial, and a size for binomial. It is unused by binary, exponential, poisson and no_zero_poisson; for uniform and uniform_integer formula instead carries both limits, and for categorical formula carries the category probabilities and variance their labels. A marginal outside those twelve raises <code>simulab_no_quantile</code> ; state it as a distribution call instead.
rho	Optional latent Pearson correlation, zero by default.
tau	Optional Kendall correlation, overriding rho. It is converted to the latent Pearson correlation $\sin(\pi * \tau / 2)$, so a continuous margin pair attains Kendall's tau of tau.
structure	Correlation structure: one of "independent", "exchangeable", "ar1", or "custom". If left unset it is chosen from the other arguments: "custom" when correlation is supplied, "exchangeable" when a non-zero rho (or tau) is supplied, and "independent" otherwise. Passing "independent" together with a non-zero rho is a contradiction and raises an error.

correlation	Custom latent correlation matrix or tidy table.
id	Identifier name.
seed	Optional random seed.
envir	Formula evaluation environment.
batch	Optional single positive whole number. When given, the simulator runs batch times and returns a plain list of batch results, each exactly what the same call without batch returns. With a seed, every dataset gets its own seed drawn from seed, so the whole batch is reproducible; without one the datasets are consecutive draws from the session's random-number stream. The default NULL returns a single result.

Value

A `simulab_sim` base data.frame with one row per observation and columns `id` (renamed by `id`) and one column per variable. Two tidy tables come from `as.data.frame()`: `what = "definitions"` restates the specification, and `what = "latent_correlation"` has one row per correlation-matrix cell with columns `row`, `column` and `correlation`. That matrix is the correlation of the latent Gaussian variables, not of the generated margins, which are pulled apart by their own quantile transforms.

Conditions

`simulab_no_quantile` when a distribution call names a marginal with no quantile function, such as `rice()` or `mixture()`.

Examples

```
# Distribution calls. Each marginal keeps its own distribution while the
# variables are correlated through a Gaussian copula.
result <- simulate_copula(
  n = 200,
  specification = define_variables(
    score = normal(mean = 10, sd = 2),
    visits = poisson(lambda = 4),
    rate = beta(shape1 = 2, shape2 = 5)
  ),
  rho = 0.5, seed = 1
)
head(result)

specification <- define_variables(
  define_variable("normal_var", formula = "0", variance = "1", distribution = "normal"),
  define_variable("count_var", formula = "3", variance = "0", distribution = "poisson")
)
result <- simulate_copula(n = 200, specification = specification, rho = 0.5, seed = 1)
head(result)
components(result)
```

simulate_correlated *Simulate correlated Gaussian variables*

Description

Simulate correlated Gaussian variables

Usage

```
simulate_correlated(
  n,
  means,
  sds = 1,
  rho = 0,
  structure = c("independent", "exchangeable", "ar1", "custom"),
  correlation = NULL,
  variable_names = NULL,
  seed = NULL,
  batch = NULL
)
```

Arguments

n	Number of observations.
means	Variable means.
sds	Variable standard deviations.
rho	Correlation used when correlation is not supplied.
structure	Correlation structure: one of "independent", "exchangeable", "ar1", or "custom". If left unset it is chosen from the other arguments: "custom" when correlation is supplied, "exchangeable" when a non-zero rho is supplied, and "independent" otherwise. Passing "independent" together with a non-zero rho is a contradiction and raises a <code>simulab_contradictory_structure</code> error.
correlation	A custom correlation matrix or tidy table returned by <code>correlation_structure()</code> .
variable_names	Optional variable names.
seed	Optional random seed.
batch	Optional single positive whole number. When given, the simulator runs batch times and returns a plain list of batch results, each exactly what the same call without batch returns. With a seed, every dataset gets its own seed drawn from seed, so the whole batch is reproducible; without one the datasets are consecutive draws from the session's random-number stream. The default NULL returns a single result.

Value

A `simulab_sim` base `data.frame` with one row per observation and columns `id` and one column per variable. Three tidy tables come from `as.data.frame()`: `what = "parameters"` has one row per variable with columns `variable`, `mean` and `sd`; `what = "correlation"` and `what = "covariance"` each have one row per matrix cell with columns `row`, `column` and `correlation` or `covariance`. The covariance is the correlation matrix scaled by `sds` on both sides, so `sds` are standard deviations rather than variances.

Examples

```
simulate_correlated(  
  n = 100,  
  means = c(0, 1, 2),  
  sds = c(1, 2, 1),  
  rho = 0.4,  
  structure = "exchangeable",  
  seed = 1  
)
```

<code>simulate_correlation</code>	<i>Simulate a correlation design</i>
-----------------------------------	--------------------------------------

Description

A thin wrapper around `simulate_correlated()` that supplies defaults for `means` and `sds`, so `simulate_correlation(n)` generates a single standard normal variable. `structure` is forwarded only when it is supplied, which keeps `simulate_correlated()`'s own default resolution intact: "custom" when correlation is given, "exchangeable" when a non-zero `rho` is given, and "independent" otherwise.

Usage

```
simulate_correlation(  
  n,  
  means = 0,  
  sds = 1,  
  rho = 0,  
  structure = c("independent", "exchangeable", "ar1", "custom"),  
  correlation = NULL,  
  variable_names = NULL,  
  seed = NULL,  
  batch = NULL  
)
```

Arguments

`n` Number of observations.

`means, sds, rho, structure, correlation, variable_names, seed`
 Arguments passed to `simulate_correlated()`. Unlike that function, `means` defaults to `0`. One variable is generated per element of `means`; `sds` must be a single value, which is recycled, or one value per mean.

`batch` Optional single positive whole number. When given, the simulator runs `batch` times and returns a plain list of batch results, each exactly what the same call without `batch` returns. With a `seed`, every dataset gets its own seed drawn from `seed`, so the whole batch is reproducible; without one the datasets are consecutive draws from the session's random-number stream. The default `NULL` returns a single result.

Value

A `simulab_sim` base data.frame with one row per observation and an `id` column followed by one column per variable. The `parameters`, `correlation` and `covariance` components hold the requested means and standard deviations and the tidy correlation and covariance matrices; use `components()` to list them.

Examples

```
result <- simulate_correlation(n = 200, means = c(0, 0), sds = c(1, 1), rho = 0.6,
                              seed = 1)
head(result)
as.data.frame(result, what = "correlation")
```

simulate_data	<i>Simulate data through the unified catalogue</i>
---------------	--

Description

`simulate_data()` is a discoverable dispatcher. Direct verbs remain the preferred interface because they provide explicit, documented arguments.

Usage

```
simulate_data(type, ...)
```

Arguments

`type` Dispatchable simulator name, taken from the `simulator` column of `list_simulators()`. An unknown name, or a name whose dispatchable entry is `FALSE`, is an error.

`...` Arguments passed on to the selected canonical simulation verb, including `batch` for every verb that produces one dataset.

Value

Whatever the selected verb returns, which for every dispatchable entry is a `simulab_sim` base `data.frame`, or a plain list of them when `batch` is given.

Examples

```
result <- simulate_data("ttest", n_a = 30, n_b = 30, mean_a = 0, mean_b = 0.5, seed = 1)
head(result)
# Five independent datasets from one call.
datasets <- simulate_data("ttest", n_a = 30, n_b = 30, mean_a = 0,
                           mean_b = 0.5, seed = 1, batch = 5)
length(datasets)
```

`simulate_density`

Simulate from an empirical kernel density

Description

Simulate from an empirical kernel density

Usage

```
simulate_density(
  n,
  values,
  variable = "value",
  use_limits = FALSE,
  keep_missing = FALSE,
  id = "id",
  seed = NULL,
  batch = NULL
)
```

Arguments

<code>n</code>	Number of observations.
<code>values</code>	Numeric source values, of which at least two must be present and distinct. Missing values are dropped before the density is estimated.
<code>variable</code>	Output variable name.
<code>use_limits</code>	Constrain draws to the observed range.
<code>keep_missing</code>	Draw missing values at the source's missing-data rate, so the proportion is reproduced in expectation rather than exactly.
<code>id</code>	Identifier variable name.
<code>seed</code>	Optional random seed.

batch Optional single positive whole number. When given, the simulator runs batch times and returns a plain list of batch results, each exactly what the same call without batch returns. With a seed, every dataset gets its own seed drawn from seed, so the whole batch is reproducible; without one the datasets are consecutive draws from the session's random-number stream. The default NULL returns a single result.

Value

A `simulab_sim` base `data.frame` with `n` rows and columns `id` and `variable`, one generated density value per row. `as.data.frame(x, what = "source")` gives a one-row summary of the source values.

Examples

```
values <- stats::rnorm(200, mean = 5, sd = 2)
result <- simulate_density(n = 100, values = values, variable = "score", seed = 1)
head(result)
```

<code>simulate_edge_list</code>	<i>Simulate a tidy network edge list</i>
---------------------------------	--

Description

Simulate a tidy network edge list

Usage

```
simulate_edge_list(
  nodes,
  edges = NULL,
  probability = 0.1,
  directed = TRUE,
  loops = FALSE,
  weight = c("binary", "uniform", "normal", "poisson"),
  weight_mean = 1,
  weight_sd = 1,
  weight_range = c(0.1, 1),
  node_type = NULL,
  edge_classes = NULL,
  class_probabilities = NULL,
  seed = NULL,
  batch = NULL
)
```

Arguments

nodes	Number of nodes or unique node labels.
edges	Optional exact number of edges.
probability	Edge probability when edges is NULL.
directed	Generate directed edges.
loops	Permit self-loops.
weight	Edge-weight distribution.
weight_mean, weight_sd, weight_range	Weight parameters.
node_type	Optional node type for each node.
edge_classes	Optional number or labels of edge classes.
class_probabilities	Optional edge-class probabilities.
seed	Optional seed.
batch	Optional single positive whole number. When given, the simulator runs batch times and returns a plain list of batch results, each exactly what the same call without batch returns. With a seed, every dataset gets its own seed drawn from seed, so the whole batch is reproducible; without one the datasets are consecutive draws from the session's random-number stream. The default NULL returns a single result.

Value

A tidy `simulab_sim` base `data.frame` with one row per edge and columns `from`, `to`, and `weight` (plus `edge_class` when `edge_classes` is supplied). `as.data.frame(x, what = )` also returns `nodes` (`node`, `type`), `adjacency` (a long row/column/weight table over all ordered node pairs), and a one-row settings table.

Examples

```
result <- simulate_edge_list(nodes = 40, probability = 0.08, seed = 1)
head(result)
components(result)
```

<code>simulate_event_log</code>	<i>Simulate grouped educational event logs</i>
---------------------------------	--

Description

Simulate grouped educational event logs

Usage

```
simulate_event_log(
  groups = 5L,
  actors = 10L,
  courses = 1L,
  states = NULL,
  n_states = 8L,
  state_categories = "all",
  sequence_length = c(10L, 30L),
  achievement_levels = c("low", "medium", "high"),
  achievement_probabilities = NULL,
  start_time = as.POSIXct("2020-01-01", tz = "UTC"),
  interval_range = c(60, 600),
  transitions = NULL,
  initial = NULL,
  seed = NULL,
  ...,
  batch = NULL
)
```

Arguments

<code>groups</code>	Number of groups.
<code>actors</code>	Actors per group.
<code>courses</code>	Number or labels of courses.
<code>states, n_states, state_categories</code>	State-space options.
<code>sequence_length</code>	Fixed length or minimum/maximum range.
<code>achievement_levels</code>	Achievement labels.
<code>achievement_probabilities</code>	Achievement probabilities, one per level and summing to one. When NULL, the levels are equally likely.
<code>start_time</code>	Initial timestamp. Every actor's first event is placed at this time.
<code>interval_range</code>	Minimum and maximum seconds between an actor's consecutive events, drawn uniformly.
<code>transitions</code>	Common matrix, one matrix per group, or NULL.
<code>initial</code>	Initial probabilities.
<code>seed</code>	Optional random seed.
<code>...</code>	Advanced options passed to <code>simulate_group_sequences()</code> .
<code>batch</code>	Optional single positive whole number. When given, the simulator runs batch times and returns a plain list of batch results, each exactly what the same call without batch returns. With a seed, every dataset gets its own seed drawn from seed, so the whole batch is reproducible; without one the datasets are consecutive draws from the session's random-number stream. The default NULL returns a single result.

Value

A long-form `simulab_sim` base `data.frame` with one row per actor event and columns `group`, `id`, `course`, `achievement`, `period`, `state`, and `timestamp`. `as.data.frame(x, what = )` also returns transitions (`group`, `from`, `to`, `probability`), actors (`group`, `id`, `course`, `achievement`), groups (`group`, `actors`), wide (one row per actor, columns `id`, `S1`, ..., plus actor metadata), and `one_hot` (the event log with state replaced by indicator columns).

Examples

```
result <- simulate_event_log(groups = 2, actors = 10, sequence_length = 8, seed = 1)
head(result)
components(result)
```

<code>simulate_factors</code>	<i>Simulate a common-factor model</i>
-------------------------------	---------------------------------------

Description

Draws factor scores from a multivariate normal with unit variances and correlation `factor_correlation`, then forms each indicator as `intercept + loadings %*% scores + error`, where the error standard deviation is `sqrt(uniqueness)`. The implied population covariance is `loadings %*% factor_correlation %*% t(loadings) + diag(uniquenesses)`. Because the factors have unit variance, the loadings are on the standardized-factor metric.

Usage

```
simulate_factors(
  n,
  loadings,
  uniquenesses = NULL,
  factor_correlation = NULL,
  intercepts = 0,
  include_scores = FALSE,
  seed = NULL,
  batch = NULL
)
```

Arguments

<code>n</code>	Number of observations. A single whole number of at least 2.
<code>loadings</code>	Variable-by-factor loading matrix (one row per indicator, one column per factor, at least 2 rows), or a tidy data frame with columns <code>item</code> , <code>factor</code> and <code>loading</code> . Row and column names, when present, name the indicators and the factors.
<code>uniquenesses</code>	Residual <i>variances</i> (not standard deviations), a scalar recycled across indicators or one non-negative value per indicator. <code>NULL</code> (the default) uses <code>1 - rowSums(loadings^2)</code> , which gives every indicator unit total variance, so the loadings are then fully standardized. Loadings whose squared row sum exceeds one therefore raise an error rather than producing a Heywood case.

factor_correlation	Optional factor correlation matrix, or a tidy data frame with columns row, column and correlation. NULL (the default) makes the factors orthogonal.
intercepts	Variable intercepts, a scalar recycled across indicators (the default, 0) or one value per indicator.
include_scores	Single flag; when TRUE the true factor scores are appended to the primary data as one column per factor. Defaults to FALSE.
seed	Optional random seed.
batch	Optional single positive whole number. When given, the simulator runs batch times and returns a plain list of batch results, each exactly what the same call without batch returns. With a seed, every dataset gets its own seed drawn from seed, so the whole batch is reproducible; without one the datasets are consecutive draws from the session's random-number stream. The default NULL returns a single result.

Value

A `simulab_sim` base data.frame with one row per observation and columns `id` and one per indicator, plus one column per factor when `include_scores = TRUE`. A `parameters` component holds one row per variable-by-factor combination with columns `variable`, `factor`, `loading`, `uniqueness` and `intercept`; a `covariance` component holds the implied population covariance in tidy form, one row per variable pair, with columns `row`, `column` and `covariance`.

Examples

```
# `loadings` is one row per item and one column per factor.
loadings <- matrix(
  c(0.8, 0.7, 0.6, 0, 0, 0,
    0, 0, 0, 0.8, 0.7, 0.6),
  ncol = 2
)
result <- simulate_factors(n = 300, loadings = loadings, seed = 1)
head(result)
as.data.frame(result, what = "parameters")
```

simulate_group_sequences

Simulate grouped actor sequences

Description

Simulate grouped actor sequences

Usage

```
simulate_group_sequences(
  groups = 2L,
  actors = 50L,
  transitions = NULL,
  chain_length = 20L,
  initial = NULL,
  group_names = NULL,
  states = NULL,
  n_states = 5L,
  state_categories = c("metacognitive", "cognitive"),
  seed = NULL,
  ...,
  batch = NULL
)
```

Arguments

<code>groups</code>	Number of groups, defaulting to 2.
<code>actors</code>	Actors per group, scalar or one value per group, defaulting to 50.
<code>transitions</code>	A common transition matrix, one matrix per group, or NULL for randomly generated matrices. A tidy data frame with columns <code>group</code> , <code>from</code> , <code>to</code> and <code>probability</code> gives one matrix per group.
<code>chain_length</code>	Sequence length, defaulting to 20.
<code>initial</code>	Common initial probabilities, one vector per group, or NULL. A tidy data frame with columns <code>group</code> , <code>state</code> and <code>probability</code> gives one vector per group.
<code>group_names</code>	Optional group labels.
<code>states</code> , <code>n_states</code> , <code>state_categories</code>	State-space arguments passed to <code>simulate_sequences()</code> .
<code>seed</code>	Optional random seed. Group-specific deterministic offsets are used without leaking RNG state.
<code>...</code>	Advanced sequence arguments passed to <code>simulate_sequences()</code> .
<code>batch</code>	Optional single positive whole number. When given, the simulator runs <code>batch</code> times and returns a plain list of batch results, each exactly what the same call without <code>batch</code> returns. With a <code>seed</code> , every dataset gets its own seed drawn from <code>seed</code> , so the whole batch is reproducible; without one the datasets are consecutive draws from the session's random-number stream. The default NULL returns a single result.

Value

A long-form `simulab_sim` base data.frame with one row per actor sequence position and columns `group`, `id`, `period`, and `state`. Actor identifiers are made unique across groups as `G<group>_A<actor>`. `as.data.frame(x, what = )` also returns transitions (`group`, `from`, `to`, `probability`), wide (one row per actor: `id`, `S1`, `S2`, ..., `group`), and groups (`group`, `actors`).

Examples

```
result <- simulate_group_sequences(
  groups = 2, actors = 20, chain_length = 12, n_states = 3, seed = 1
)
head(result)
components(result)
```

simulate_group_tna	<i>Simulate sequences and fit a grouped TNA model</i>
--------------------	---

Description

Simulate sequences and fit a grouped TNA model

Usage

```
simulate_group_tna(
  groups = 2L,
  actors = 50L,
  transitions = NULL,
  chain_length = 20L,
  initial = NULL,
  group_names = NULL,
  states = NULL,
  n_states = 5L,
  state_categories = c("metacognitive", "cognitive"),
  model = c("tna", "ftna", "ctna", "atna"),
  seed = NULL,
  ...,
  batch = NULL
)
```

Arguments

groups	Number of groups, defaulting to 2.
actors	Actors per group, scalar or one value per group, defaulting to 50.
transitions	A common transition matrix, one matrix per group, or NULL for randomly generated matrices. A tidy data frame with columns group, from, to and probability gives one matrix per group.
chain_length	Sequence length, defaulting to 20.
initial	Common initial probabilities, one vector per group, or NULL. A tidy data frame with columns group, state and probability gives one vector per group.
group_names	Optional group labels.
states, n_states, state_categories	State-space arguments passed to simulate_sequences().

model	TNA model type.
seed	Optional random seed. Group-specific deterministic offsets are used without leaking RNG state.
...	Advanced sequence arguments.
batch	Optional single positive whole number. When given, the simulator runs batch times and returns a plain list of batch results, each exactly what the same call without batch returns. With a seed, every dataset gets its own seed drawn from seed, so the whole batch is reproducible; without one the datasets are consecutive draws from the session's random-number stream. The default NULL returns a single result.

Value

A long-form `simulab_sim` base data.frame of grouped sequences with columns `group`, `id`, `period`, and `state`. `as.data.frame(x, what = )` also returns `true_transitions` (the generating probabilities), `estimated_edges` (the fitted TNA weights), `model_info`, `wide`, and `groups`. The native `group_tna` model is accessible through `as_tna_model()`. Requires the suggested `tna` package.

Examples

```
if (requireNamespace("tna", quietly = TRUE)) {
  result <- simulate_group_tna(
    groups = 2, actors = 20, chain_length = 12, n_states = 3, seed = 1
  )
  head(result)
  components(result)

  # With no arguments, one transition matrix per group is drawn and fitted.
  drawn <- simulate_group_tna(seed = 1)
  as.data.frame(drawn, what = "true_transitions")
}
```

simulate_growth

Simulate longitudinal growth trajectories

Description

Draws per-unit random growth coefficients, then forms the outcome at each measurement occasion as $(\text{intercept} + b_0) + (\text{slope} + b_1) * \text{time} + (\text{quadratic} + b_2) * \text{time}^2$ plus normal residual noise, where `b2` is present only when `random_sd` has three elements. Every unit is observed at every occasion (balanced design).

Usage

```
simulate_growth(
  n,
  times,
```

```

    intercept = 0,
    slope = 1,
    quadratic = 0,
    random_sd = c(1, 0.25),
    random_correlation = NULL,
    residual_sd = 1,
    seed = NULL,
    batch = NULL
  )

```

Arguments

<code>n</code>	Number of units. A single whole number of at least 2.
<code>times</code>	Measurement occasions: a finite numeric vector of at least two unique values, used verbatim as the time predictor (so <code>0:4</code> puts the intercept at the first occasion).
<code>intercept, slope, quadratic</code>	Fixed growth coefficients, each a single number; they default to 0, 1 and 0.
<code>random_sd</code>	Standard deviations for the random intercept, slope, and optional quadratic term. A non-negative numeric vector of length 2 (the default, <code>c(1, 0.25)</code>) or 3.
<code>random_correlation</code>	Correlation matrix for the random effects, matching the length of <code>random_sd</code> , or a tidy data frame with columns <code>row</code> , <code>column</code> and <code>correlation</code> . <code>NULL</code> (the default) makes the random effects uncorrelated.
<code>residual_sd</code>	Residual standard deviation. A single positive number, defaulting to 1.
<code>seed</code>	Optional random seed.
<code>batch</code>	Optional single positive whole number. When given, the simulator runs <code>batch</code> times and returns a plain list of batch results, each exactly what the same call without <code>batch</code> returns. With a <code>seed</code> , every dataset gets its own seed drawn from <code>seed</code> , so the whole batch is reproducible; without one the datasets are consecutive draws from the session's random-number stream. The default <code>NULL</code> returns a single result.

Value

A long-form `simulab_sim` base data.frame with one row per unit-occasion (`n * length(times)` rows) and columns `id`, `time` and `outcome`. Rows are ordered by occasion first and unit second. A `parameters` component holds one row for each of `intercept`, `slope`, `quadratic` and `residual_sd`, with columns `term` and `value` (note that `residual_sd` is a standard deviation, and that the random-effect standard deviations are not repeated there). A `random_effects` component holds one row per unit with columns `id`, `intercept`, `slope` and, when `random_sd` has three elements, `quadratic`, giving each unit's deviation from the fixed coefficients.

Examples

```

result <- simulate_growth(
  n = 100, times = 0:4, intercept = 10, slope = 0.5, seed = 1
)

```

```
head(result)
as.data.frame(result, what = "parameters")
```

simulate_hmm

Simulate a hidden Markov model

Description

Draws an initial hidden state for each sequence from `initial`, evolves it with transition for `chain_length` occasions, and emits one observed category per occasion from `emission`.

Usage

```
simulate_hmm(
  n,
  transition,
  chain_length,
  emission,
  initial = NULL,
  state_labels = NULL,
  observation_labels = NULL,
  seed = NULL,
  batch = NULL
)
```

Arguments

<code>n</code>	Number of sequences. A single positive whole number.
<code>transition</code>	Hidden-state transition matrix, or a tidy data frame with columns <code>from</code> , <code>to</code> and <code>probability</code> . Square, with rows indexing the current state and columns the next state; every row must sum to one.
<code>chain_length</code>	Sequence length. A single positive whole number; every sequence has the same length.
<code>emission</code>	Hidden-state-by-observed-category probability matrix, or a tidy data frame with columns <code>state</code> , <code>observation</code> and <code>probability</code> . Rows index the hidden states and must sum to one; columns index the observed categories.
<code>initial</code>	Initial hidden-state probabilities, one per hidden state, summing to one. <code>NULL</code> (the default) starts every sequence in the first hidden state with probability one; supply a vector for a random start.
<code>state_labels, observation_labels</code>	Optional labels, one unique value per hidden state and per observed category. They default to "State 1", "State 2", ... and "Observation 1", "Observation 2", ...
<code>seed</code>	Optional random seed.

batch Optional single positive whole number. When given, the simulator runs batch times and returns a plain list of batch results, each exactly what the same call without batch returns. With a seed, every dataset gets its own seed drawn from seed, so the whole batch is reproducible; without one the datasets are consecutive draws from the session's random-number stream. The default NULL returns a single result.

Value

A long-form `simulab_sim` base data.frame with one row per sequence-occasion ($n \times \text{chain_length}$ rows) and columns `id`, `occasion`, `state` (the true hidden state label) and `observation` (the emitted category label). Components `transitions` (columns `from`, `to`, `probability`), `emissions` (columns `state`, `observation`, `probability`) and `initial_probabilities` (columns `state`, `probability`) hold the generating parameters in tidy form.

Examples

```
result <- simulate_hmm(
  n = 50,
  transition = matrix(c(0.7, 0.3, 0.4, 0.6), nrow = 2, byrow = TRUE),
  chain_length = 20,
  emission = matrix(c(0.9, 0.1, 0.2, 0.8), nrow = 2, byrow = TRUE),
  seed = 1
)
head(result)
components(result)
```

<code>simulate_irt</code>	<i>Simulate item-response data</i>
---------------------------	------------------------------------

Description

Draws latent abilities from a standard multivariate normal (mean zero, unit variance, correlation `ability_correlation`), projects them onto each item through dimensions, and samples responses from a logistic item-response model.

Usage

```
simulate_irt(
  n,
  discrimination = 1,
  difficulty,
  dimensions = NULL,
  ability_correlation = NULL,
  model = c("2pl", "rasch", "3pl", "graded"),
  guessing = 0.2,
  seed = NULL,
  batch = NULL
)
```

Arguments

n	Number of respondents. A single whole number of at least 2.
discrimination	Positive item discriminations, a scalar recycled across items (the default, 1) or one value per item. Must equal one for model = "rasch".
difficulty	Item difficulties, on the same scale as the latent ability. A numeric vector, one difficulty per item, for the dichotomous models. model = "graded" instead requires an item-by-threshold matrix (or a tidy data frame with columns item, dimension and difficulty) whose thresholds increase within each item.
dimensions	Item-by-dimension loading weights, or a tidy data frame with columns item, dimension and loading. NULL (the default) uses one dimension. Each row is rescaled to unit length before use, so only the relative weights within an item matter, not their scale. Row and column names, when present, name the items and the ability columns.
ability_correlation	Latent ability correlation matrix, or a tidy data frame with columns row, column and correlation. NULL (the default) makes the abilities uncorrelated.
model	Logistic model: "2pl" (the default), "rasch", "3pl", or "graded".
guessing	Lower-asymptote guessing parameters in $[0, 1)$, a scalar recycled across items (the default, 0.2) or one value per item. It is silently reset to zero unless model = "3pl".
seed	Optional random seed.
batch	Optional single positive whole number. When given, the simulator runs batch times and returns a plain list of batch results, each exactly what the same call without batch returns. With a seed, every dataset gets its own seed drawn from seed, so the whole batch is reproducible; without one the datasets are consecutive draws from the session's random-number stream. The default NULL returns a single result.

Details

For the dichotomous models the probability of a correct response is $\text{guessing} + (1 - \text{guessing}) * \text{plogis}(\text{discrimination} * (\text{theta} - \text{difficulty}))$, so difficulty is on the ability scale (the point of inflection), not an intercept. "rasch" fixes all discriminations at one and "2pl" sets guessing to zero; only "3pl" uses guessing.

For model = "graded" the cumulative category probabilities are $\text{plogis}(\text{discrimination} * (\text{theta} - \text{threshold}))$ for each threshold of an item, and the category probabilities are their successive differences, so the thresholds are also on the ability scale.

Value

A `simulab_sim` base data.frame with one row per respondent and columns `id` and one integer column per item, coded 0/1 for the dichotomous models and 0 to the number of thresholds for model = "graded". An `abilities` component holds one row per respondent with columns `id` and one true ability per dimension. A `parameters` component holds one row per item for the dichotomous models, with columns `item`, `parameter` ("difficulty"), `category` (NA), `value`, `discrimination` and `guessing`; for model = "graded" it holds one row per item-by-threshold combination, with `parameter` equal to "threshold", `category` the threshold index, and no `guessing` column.

Examples

```

result <- simulate_irt(
  n = 200,
  discrimination = c(1, 1.2, 0.8),
  difficulty = c(-1, 0, 1),
  seed = 1
)
head(result)
as.data.frame(result, what = "parameters")

# Three-parameter logistic with a guessing floor.
head(simulate_irt(
  n = 200,
  discrimination = c(1, 1.2, 0.8),
  difficulty = c(-1, 0, 1),
  model = "3pl",
  guessing = 0.2,
  seed = 1
))

```

simulate_lca

*Simulate a latent class model***Description**

Draws a latent class for each observation from proportions, then draws every indicator independently from that class's category probabilities (the usual local-independence assumption of latent class analysis).

Usage

```

simulate_lca(
  n,
  probabilities,
  proportions = NULL,
  class_labels = NULL,
  category_labels = NULL,
  seed = NULL,
  batch = NULL
)

```

Arguments

n	Number of observations. A single whole number of at least 2.
probabilities	Item probabilities as an array with dimensions class, indicator and category, or as a tidy data frame with columns class, indicator, category and probability. Category probabilities must sum to one within each class and indicator. The dimnames (the class, indicator and category values of a tidy table) name the classes, the indicator columns and the categories.

proportions	Latent-class proportions, one positive value per class; they are rescaled to sum to one. NULL (the default) makes every class equally likely.
class_labels, category_labels	Optional labels, one unique value per class and per category, overriding the names given in probabilities. Without names there, class_labels defaults to "Class 1", "Class 2" and so on, and category_labels to the integers 1:categories. The category labels are the values written into the indicator columns.
seed	Optional random seed.
batch	Optional single positive whole number. When given, the simulator runs batch times and returns a plain list of batch results, each exactly what the same call without batch returns. With a seed, every dataset gets its own seed drawn from seed, so the whole batch is reproducible; without one the datasets are consecutive draws from the session's random-number stream. The default NULL returns a single result.

Value

A `simulab_sim` base data.frame with one row per observation and columns `id`, `latent_class` (the true class label) and one column per indicator holding the sampled value of `category_labels`. A `parameters` component holds one row per class-by-indicator-by-category combination with columns `latent_class`, `indicator`, `category`, `probability` and `proportion`.

Examples

```
# `probabilities` is a class x indicator x category array. Values fill the
# class dimension fastest, so each pair below is one category across the two
# classes: P(indicator = 1) then P(indicator = 2).
probabilities <- array(
  c(0.8, 0.2, 0.7, 0.3, # category 1, for each class and indicator
    0.2, 0.8, 0.3, 0.7), # category 2, for each class and indicator
  dim = c(2, 2, 2),
  dimnames = list(
    class = c("Class 1", "Class 2"),
    indicator = c("item_1", "item_2"),
    category = c("no", "yes")
  )
)

# Only the `indicator` dimnames name the output columns, so the class and
# category labels are passed explicitly.
result <- simulate_lca(
  n = 200,
  probabilities = probabilities,
  proportions = c(0.6, 0.4),
  category_labels = c("no", "yes"),
  seed = 1
)
head(result)
as.data.frame(result, what = "parameters")
```

simulate_longitudinal *Simulate a multivariate longitudinal VAR process*

Description

Draws a person mean for each unit from between_covariance around grand_means, then runs a first-order vector autoregression on the deviations from that person mean: $y_t = \mu + \text{intercept} + \text{transition} \% \% (y_{t-1} - \mu)$ with e_t drawn from innovation_covariance. The first burn_in occasions are discarded so the retained occasions are near stationarity.

Usage

```
simulate_longitudinal(
  n,
  occasions,
  transition,
  intercept = 0,
  innovation_covariance = NULL,
  initial_covariance = NULL,
  between_covariance = NULL,
  grand_means = 0,
  beeps_per_day = NULL,
  burn_in = 50L,
  seed = NULL,
  batch = NULL
)
```

Arguments

n	Number of units. A single positive whole number.
occasions	Number of occasions retained per unit. A single whole number of at least 2.
transition	Lag-one coefficients as a square matrix, or as a tidy data frame with columns from, to and coefficient (from supplies the matrix row and to the column). Entry [i, j] multiplies variable j at the previous occasion when forming variable i at the current occasion, so rows index the <i>receiving</i> variable and columns the <i>predicting</i> variable. The spectral radius must be below one.
intercept	Variable intercepts, a scalar recycled across variables (the default, 0) or one value per variable. The intercept is added to the deviation from the person mean at every occasion, so a non-zero intercept shifts the stationary mean to $\text{person_mean} + \text{solve}(\text{diag}(p) - \text{transition}) \% \% \text{intercept}$ and the person_means component is then no longer the realized process mean.
innovation_covariance	Innovation covariance matrix, or a tidy data frame with columns row, column and covariance. Must be symmetric and positive semidefinite. NULL (the default) uses the identity.

<code>initial_covariance</code>	Initial-state covariance matrix, or a tidy data frame with columns <code>row</code> , <code>column</code> and <code>covariance</code> . NULL (the default) reuses <code>innovation_covariance</code> .
<code>between_covariance</code>	Between-unit covariance of person means, or a tidy data frame with columns <code>row</code> , <code>column</code> and <code>covariance</code> . NULL (the default) is a zero matrix, giving every unit the same person mean, <code>grand_means</code> .
<code>grand_means</code>	Population means, a scalar recycled across variables (the default, 0) or one value per variable.
<code>beeps_per_day</code>	Optional number of occasions per day; occasions must be divisible by it. Adds <code>day</code> and <code>beep</code> columns to the returned data. Temporal carryover resets at each day boundary.
<code>burn_in</code>	Warm-up occasions generated and then discarded before output. A single non-negative whole number, defaulting to 50.
<code>seed</code>	Optional random seed.
<code>batch</code>	Optional single positive whole number. When given, the simulator runs batch times and returns a plain list of batch results, each exactly what the same call without batch returns. With a seed, every dataset gets its own seed drawn from seed, so the whole batch is reproducible; without one the datasets are consecutive draws from the session's random-number stream. The default NULL returns a single result.

Value

A long-form `simulab_sim` base data.frame with one row per unit-occasion (`n * occasions` rows) and columns `id`, `occasion` and one per variable, plus `day` and `beep` when `beeps_per_day` is supplied. Components: `wide` (one row per unit, one `<variable>.<occasion>` column per measurement), `transition`, `innovation_covariance` and `between_covariance` (tidy matrix tables with columns `row`, `column` and `coefficient` or `covariance`), and `person_means` (one row per unit with columns `id` and one per variable).

Examples

```
# `transition` is the lag-one coefficient matrix of the VAR process.
result <- simulate_longitudinal(
  n = 20,
  occasions = 30,
  transition = matrix(c(0.5, 0.1, 0.0, 0.4), nrow = 2, byrow = TRUE),
  seed = 1
)
head(result)
components(result)
```

simulate_lpa	<i>Simulate a latent profile model</i>
--------------	--

Description

Draws each observation from a multivariate normal mixture: a profile is sampled from proportions, then the indicators are drawn from that profile's means, standard deviations and within-profile correlation matrix.

Usage

```
simulate_lpa(
  n,
  means,
  sds = 1,
  proportions = NULL,
  correlations = NULL,
  labels = NULL,
  seed = NULL,
  batch = NULL
)
```

Arguments

n	Number of observations. A single whole number of at least 2.
means	Profile-by-variable mean matrix (one row per profile, one column per indicator, at least 2 rows), or a tidy data frame with columns profile, variable and mean. Column names become the indicator columns of the returned data; row names, or the profile values of a tidy table, name the profiles unless labels is given.
sds	Scalar (the default, 1), per-variable vector, profile-by-variable matrix, or a tidy data frame with columns profile, variable and sd. All values must be positive.
proportions	Profile proportions, one positive value per profile; they are rescaled to sum to one. NULL (the default) makes every profile equally likely.
correlations	Optional list of within-profile correlation matrices, one per profile, or a tidy data frame with columns profile, row, column and correlation. NULL (the default) makes the indicators uncorrelated within every profile.
labels	Optional profile labels, one unique value per profile. Defaults to the profile names given with means (the profile values of a tidy table, or the row names of a matrix), else "Profile 1", "Profile 2" and so on.
seed	Optional random seed.
batch	Optional single positive whole number. When given, the simulator runs batch times and returns a plain list of batch results, each exactly what the same call without batch returns. With a seed, every dataset gets its own seed drawn from seed, so the whole batch is reproducible; without one the datasets are

consecutive draws from the session's random-number stream. The default NULL returns a single result.

Value

A `simulab_sim` base `data.frame` with one row per observation and columns `id`, `profile` (the true profile label) and one numeric column per indicator. A `parameters` component holds one row per profile-by-variable combination with columns `profile`, `variable`, `mean`, `sd` and `proportion`.

Examples

```
# `means` is one row per profile and one column per indicator.
result <- simulate_lpa(
  n = 200,
  means = matrix(c(0, 0, 3, 3), nrow = 2, byrow = TRUE),
  proportions = c(0.6, 0.4),
  seed = 1
)
head(result)
as.data.frame(result, what = "parameters")
```

simulate_markov

Simulate Markov chains

Description

Simulate Markov chains

Usage

```
simulate_markov(
  n,
  transition,
  chain_length,
  initial = NULL,
  states = NULL,
  trim_state = NULL,
  id = "id",
  period = "period",
  state = "state",
  seed = NULL,
  batch = NULL
)
```

Arguments

<code>n</code>	Number of chains.
<code>transition</code>	Square transition matrix, or a tidy transition table with from, to, and probability columns. Every row of the implied matrix must sum to one.
<code>chain_length</code>	Chain length.
<code>initial</code>	Starting-state probabilities or a single fixed start state. When NULL, every chain starts in the first state.
<code>states</code>	Optional state labels.
<code>trim_state</code>	Optional terminal state. The first occurrence is kept and all later observations of that chain are removed.
<code>id, period, state</code>	Output variable names.
<code>seed</code>	Optional random seed.
<code>batch</code>	Optional single positive whole number. When given, the simulator runs batch times and returns a plain list of batch results, each exactly what the same call without batch returns. With a seed, every dataset gets its own seed drawn from seed, so the whole batch is reproducible; without one the datasets are consecutive draws from the session's random-number stream. The default NULL returns a single result.

Value

A long-form `simulab_sim` base `data.frame` with columns `id`, `period`, and `state` (renamed by those arguments), one row per chain position. `as.data.frame(x, what = )` also returns transitions (from, to, probability), initial_probabilities (state, probability), and wide (one row per chain, columns `id`, `S1`, `S2`, ...).

Examples

```
transition <- matrix(c(0.7, 0.3, 0.4, 0.6), nrow = 2, byrow = TRUE)

result <- simulate_markov(
  n = 50, transition = transition, chain_length = 20,
  states = c("A", "B"), seed = 1
)
head(result)
summarize_transitions(result, normalize = TRUE)

# Transitions may also be given as a tidy from/to/probability table.
tidy_transitions <- data.frame(
  from = c("A", "A", "B", "B"),
  to = c("A", "B", "A", "B"),
  probability = c(0.7, 0.3, 0.4, 0.6)
)
head(simulate_markov(
  n = 50, transition = tidy_transitions, chain_length = 20, seed = 1
))
```

simulate_ml_lpa	<i>Simulate a two-level latent profile model</i>
-----------------	--

Description

Draws data from the nonparametric two-level mixture of Vermunt (2003). Every cluster belongs to a latent cluster class, a cluster class fixes how common each individual profile is inside its clusters, and the profiles themselves share one measurement model across every cluster. Cluster classes therefore differ in profile prevalence, not in profile means, which is what separates this design from simulating one latent profile model per cluster.

Usage

```
simulate_ml_lpa(
  clusters,
  cluster_size,
  means,
  profile_probabilities,
  sds = 1,
  cluster_class_proportions = NULL,
  correlations = NULL,
  labels = NULL,
  cluster_class_labels = NULL,
  seed = NULL,
  batch = NULL
)
```

Arguments

<code>clusters</code>	Number of clusters (level-2 units). A single whole number of at least 2.
<code>cluster_size</code>	Individuals per cluster: a single whole number recycled across clusters, or one whole number per cluster, so clusters may be unbalanced.
<code>means</code>	Profile-by-variable mean matrix (one row per profile, one column per indicator, at least 2 rows), or a tidy data frame with columns <code>profile</code> , <code>variable</code> and <code>mean</code> . Column names become the indicator columns of the returned data; row names, or the profile values of a tidy table, name the profiles unless <code>labels</code> is given.
<code>profile_probabilities</code>	Cluster-class-by-profile matrix of profile prevalences (one row per cluster class, one column per profile), or a tidy data frame with columns <code>cluster_class</code> , <code>profile</code> and <code>probability</code> . Every value must be positive and each row is rescaled to sum to one. A single row gives an ordinary latent profile model observed in clusters.
<code>sds</code>	Scalar (the default, 1), per-variable vector, profile-by-variable matrix, or a tidy data frame with columns <code>profile</code> , <code>variable</code> and <code>sd</code> . All values must be positive. The measurement model is shared by every cluster class, so standard deviations do not vary across them.

<code>cluster_class_proportions</code>	Cluster-class proportions, one positive value per cluster class; they are rescaled to sum to one. NULL (the default) makes every cluster class equally likely.
<code>correlations</code>	Optional list of within-profile correlation matrices, one per profile, or a tidy data frame with columns <code>profile</code> , <code>row</code> , <code>column</code> and <code>correlation</code> . NULL (the default) makes the indicators uncorrelated within every profile, the conditional-independence measurement model.
<code>labels</code>	Optional profile labels, one unique value per profile. Defaults to the profile names given with <code>means</code> (the profile values of a tidy table, or the row names of a matrix), else "Profile 1", "Profile 2" and so on.
<code>cluster_class_labels</code>	Optional cluster-class labels, one unique value per cluster class. Defaults to "Class 1", "Class 2" and so on.
<code>seed</code>	Optional random seed.
<code>batch</code>	Optional single positive whole number. When given, the simulator runs batch times and returns a plain list of batch results, each exactly what the same call without batch returns. With a seed, every dataset gets its own seed drawn from seed, so the whole batch is reproducible; without one the datasets are consecutive draws from the session's random-number stream. The default NULL returns a single result.

Value

A `simulab_sim` base data.frame with one row per individual (`sum(cluster_size)` rows) and the columns `id`, `cluster` (the level-2 identifier), `cluster_class` (the true cluster-class label), `profile` (the true profile label) and one numeric column per indicator. A `parameters` component holds one row per profile-by-variable combination with columns `profile`, `variable`, `mean`, `sd` and `proportion`, the profile's marginal prevalence across cluster classes. A `profile_probabilities` component holds one row per cluster-class-by-profile combination with columns `cluster_class`, `profile`, `probability` and `cluster_class_proportion`. A `clusters` component holds one row per cluster with columns `cluster`, `cluster_class` and `size`.

References

Vermunt, J. K. (2003). Multilevel latent class models. *Sociological Methodology*, 33, 213–239.
[doi:10.1111/j.00811750.2003.t01100131.x](https://doi.org/10.1111/j.00811750.2003.t01100131.x)

Examples

```
# One measurement model, two cluster classes that differ only in how common
# each profile is inside their clusters.
means <- matrix(
  c(-1.5, -1.2,
    0.0,  0.2,
    1.5,  1.2),
  nrow = 3, byrow = TRUE,
  dimnames = list(NULL, c("reading", "maths")))
)
prevalence <- matrix(
```

```

      c(0.60, 0.30, 0.10,
        0.10, 0.30, 0.60),
      nrow = 2, byrow = TRUE
    )
  result <- simulate_ml_lpa(
    clusters = 30,
    cluster_size = 20,
    means = means,
    profile_probabilities = prevalence,
    seed = 1
  )
  head(result)
  as.data.frame(result, what = "profile_probabilities")
  as.data.frame(result, what = "clusters")

```

simulate_multilevel	<i>Simulate a two-level Gaussian model</i>
---------------------	--

Description

Draws a random intercept (and optionally a random slope) per cluster, draws the predictors independently for every unit, and forms the outcome as the fixed part plus the cluster random effects plus normal residual noise.

Usage

```

simulate_multilevel(
  clusters,
  cluster_size,
  intercept = 0,
  slopes = numeric(0L),
  predictor_means = 0,
  predictor_sds = 1,
  random_intercept_sd = 1,
  random_slope_sd = 0,
  random_effect_correlation = 0,
  residual_sd = 1,
  seed = NULL,
  batch = NULL
)

```

Arguments

clusters	Number of clusters. A single whole number of at least 2.
cluster_size	Units per cluster: a single whole number recycled across clusters, or one whole number per cluster, so clusters may be unbalanced.
intercept	Fixed intercept. A single number, defaulting to 0.

slopes	Named fixed-effect slopes; the names become the predictor columns of the returned data. Defaults to <code>numeric(0)</code> , an intercept-only model with no predictor columns.
predictor_means, predictor_sds	Mean and standard deviation of each predictor, a scalar recycled across predictors (the defaults, 0 and 1) or one value per entry of slopes. Predictors are drawn independently at the unit level, not at the cluster level, so they carry no between-cluster variance by construction.
random_intercept_sd	Random-intercept standard deviation. A single non-negative number, defaulting to 1.
random_slope_sd	Random-slope standard deviation for the first predictor only; the remaining predictors have fixed slopes. A single non-negative number, defaulting to 0 (no random slope).
random_effect_correlation	Correlation between random intercept and slope. A single number in $[-1, 1]$, defaulting to 0.
residual_sd	Residual standard deviation. A single positive number, defaulting to 1.
seed	Optional random seed.
batch	Optional single positive whole number. When given, the simulator runs batch times and returns a plain list of batch results, each exactly what the same call without batch returns. With a seed, every dataset gets its own seed drawn from seed, so the whole batch is reproducible; without one the datasets are consecutive draws from the session's random-number stream. The default NULL returns a single result.

Value

A `simulab_sim` base data.frame with one row per unit (`sum(cluster_size)` rows) and columns `id`, `cluster`, one column per name of slopes, and `outcome`. Components: `fixed_effects` (one row per fixed term, columns `term` and `coefficient`), `variance_components` (one row for each of `random_intercept`, `random_slope` and `residual`, columns `component` and `variance` – the *squared* standard deviations supplied as arguments), and `random_effects` (one row per cluster, columns `cluster`, `random_intercept` and `random_slope`; the `random_slope` column is present but zero when `random_slope_sd = 0`).

Examples

```
# `slopes` must be named; the names become the predictor columns.
result <- simulate_multilevel(
  clusters = 30,
  cluster_size = 10,
  intercept = 1,
  slopes = c(x1 = 0.5),
  random_intercept_sd = 0.8,
  seed = 1
)
```

```
head(result)
as.data.frame(result, what = "fixed_effects")
as.data.frame(result, what = "variance_components")
```

```
simulate_multiplex_network
```

Simulate a multiplex network

Description

Simulate a multiplex network

Usage

```
simulate_multiplex_network(
  nodes,
  layers,
  edges = NULL,
  probability = 0.1,
  directed = TRUE,
  loops = FALSE,
  weight = c("binary", "uniform", "normal", "poisson"),
  weight_mean = 1,
  weight_sd = 1,
  weight_range = c(0.1, 1),
  node_type = NULL,
  seed = NULL,
  batch = NULL
)
```

Arguments

<code>nodes</code>	Number of nodes or unique labels shared across layers.
<code>layers</code>	Number of layers or unique layer labels.
<code>edges</code>	Optional exact edge count, scalar or one value per layer.
<code>probability</code>	Edge probability, scalar or one value per layer.
<code>directed</code>	Generate directed edges.
<code>loops</code>	Permit self-loops.
<code>weight</code>	Edge-weight distribution.
<code>weight_mean, weight_sd, weight_range</code>	Weight parameters.
<code>node_type</code>	Optional node type for each node.
<code>seed</code>	Optional base seed; layers use deterministic offsets.

batch Optional single positive whole number. When given, the simulator runs batch times and returns a plain list of batch results, each exactly what the same call without batch returns. With a seed, every dataset gets its own seed drawn from seed, so the whole batch is reproducible; without one the datasets are consecutive draws from the session's random-number stream. The default NULL returns a single result.

Value

A tidy layered edge-list `simulab_sim` base data.frame with one row per edge and columns `layer`, `from`, `to`, and `weight`. Every layer is drawn independently over the same node set. `as.data.frame(x, what = )` also returns `nodes` (`node`, `type`), `layers` (`layer`, `probability`), `adjacency` (a long layer/row/column/weight table over all ordered node pairs in every layer), and a one-row settings table.

Examples

```
result <- simulate_multiplex_network(nodes = 25, layers = 3, probability = 0.1, seed = 1)
head(result)
components(result)
```

simulate_network	<i>Simulate a network from common graph models</i>
------------------	--

Description

Simulate a network from common graph models

Usage

```
simulate_network(
  nodes,
  model = c("bernoulli", "barabasi_albert", "small_world", "block", "regular",
    "geometric", "forest_fire"),
  probability = 0.1,
  edges = NULL,
  directed = TRUE,
  loops = FALSE,
  weight = c("binary", "uniform", "normal", "poisson"),
  weight_mean = 1,
  weight_sd = 1,
  weight_range = c(0.1, 1),
  node_type = NULL,
  edge_classes = NULL,
  class_probabilities = NULL,
  attachment = 2L,
  power = 1,
  neighbors = 2L,
```

```

    rewired = 0.05,
    blocks = 3L,
    within_probability = 0.3,
    between_probability = 0.05,
    degree = 4L,
    radius = 0.25,
    forward_probability = 0.35,
    backward_probability = 0.32,
    seed = NULL,
    batch = NULL
  )

```

Arguments

nodes	Number of nodes or node labels. A single number produces the integer labels 1:nodes.
model	Graph model. Every model except <code>bernoulli</code> requires the suggested <code>igraph</code> package.
probability	Bernoulli edge probability, node matrix, or type matrix. A matrix may instead be given as a tidy data frame with columns <code>from</code> , <code>to</code> and <code>probability</code> . Used by the <code>bernoulli</code> model only; the block model uses <code>within_probability</code> and <code>between_probability</code> instead.
edges	Exact edge count for the fixed-edge Bernoulli model, capped at the number of admissible dyads.
directed	Generate directed edges. Ignored by the <code>small_world</code> and <code>geometric</code> models, which are always undirected; the <code>directed</code> column of the settings table reports what was actually generated.
loops	Permit self-loops.
weight	Edge-weight distribution.
weight_mean, weight_sd, weight_range	Weight parameters.
node_type	Optional type label for each node. With more than one type it also supplies the block membership used by the block model.
edge_classes	Optional number or labels of edge classes.
class_probabilities	Optional edge-class probabilities.
attachment, power	Preferential-attachment parameters.
neighbors, rewired	Small-world parameters.
blocks, within_probability, between_probability	Block-model parameters.
degree	Regular-graph degree.
radius	Geometric-graph connection radius.

forward_probability, backward_probability	Forest-fire parameters.
seed	Optional random seed.
batch	Optional single positive whole number. When given, the simulator runs batch times and returns a plain list of batch results, each exactly what the same call without batch returns. With a seed, every dataset gets its own seed drawn from seed, so the whole batch is reproducible; without one the datasets are consecutive draws from the session's random-number stream. The default NULL returns a single result.

Value

A tidy edge-list `simulab_sim` base `data.frame` with one row per edge and columns `from`, `to`, `weight` (plus `edge_class` when `edge_classes` is supplied). `as.data.frame(x, what = )` also returns `nodes` (`node`, `type`), `adjacency` (a long row/column/weight table covering all `nodes^2` ordered pairs, symmetric when the generated graph is undirected), and a one-row settings table. Use `as_igraph()` for native graph workflows.

Examples

```
result <- simulate_network(
  nodes = 60, model = "bernoulli", probability = 0.06, seed = 1
)
head(result)
components(result)

# Other generators: barabasi_albert, small_world, block, regular,
# geometric and forest_fire. These require the suggested igraph package.
if (requireNamespace("igraph", quietly = TRUE)) {
  head(simulate_network(nodes = 60, model = "small_world", neighbors = 2, seed = 1))
}
```

`simulate_network_batches`

Simulate repeated networks

Description

Calls `simulate_network()` repetitions times and stacks the edge lists.

Usage

```
simulate_network_batches(repetitions, ..., seed = NULL)
```

Arguments

repetitions	Number of networks. A single positive whole number.
...	Arguments passed to <code>simulate_network()</code> .
seed	Optional base seed. A single number, or NULL (the default). Network <i>i</i> uses <code>seed + i - 1</code> .

Value

A `simulab_sim` base `data.frame` edge list, one row per network and edge, with the column `network` followed by the columns `simulate_network()` returns (`from`, `to` and `weight`). The component settings, reached with `as.data.frame(x, what = "settings")`, holds one row per network with its generator arguments and realized edge count.

Examples

```
result <- simulate_network_batches(
  repetitions = 3, nodes = 20, model = "bernoulli", probability = 0.1, seed = 1
)
head(result)
```

```
simulate_network_matrix
```

Simulate a network matrix

Description

Simulate a network matrix

Usage

```
simulate_network_matrix(
  nodes,
  type = c("adjacency", "transition", "frequency", "cooccurrence"),
  probability = 0.2,
  directed = TRUE,
  loops = FALSE,
  weighted = TRUE,
  weight_range = c(0.1, 1),
  frequency_mean = 10,
  seed = NULL,
  batch = NULL
)
```

Arguments

nodes	Number of nodes or unique node labels.
type	Matrix type.
probability	Probability of a non-zero dyad.
directed	Generate an asymmetric matrix where applicable.
loops	Permit non-zero diagonal entries.
weighted	Generate weighted adjacency/co-occurrence values. Ignored by the frequency and transition types, which are always weighted.
weight_range	Range for continuous weights.
frequency_mean	Mean positive count for frequency matrices; drawn as $1 + \text{rpois}(\text{frequency_mean} - 1)$, so entries are at least one.
seed	Optional seed.
batch	Optional single positive whole number. When given, the simulator runs batch times and returns a plain list of batch results, each exactly what the same call without batch returns. With a seed, every dataset gets its own seed drawn from seed, so the whole batch is reproducible; without one the datasets are consecutive draws from the session's random-number stream. The default NULL returns a single result.

Value

A tidy `simulab_sim` base data.frame holding the complete matrix in long form: nodes^2 rows with columns `from`, `to`, and `value`, zeros included. `as.data.frame(x, what = )` also returns edges (the non-zero rows only), nodes (node), matrix (the wide form: a from column followed by one column per node), and a one-row settings table whose `directed` field reports whether the generated matrix is asymmetric. The transition type is row-stochastic: every from state's value entries sum to one.

Examples

```
result <- simulate_network_matrix(nodes = 8, type = "adjacency", seed = 1)
head(result)
components(result)
```

simulate_ordinal

Simulate correlated ordinal variables

Description

Simulate correlated ordinal variables

Usage

```
simulate_ordinal(
  n,
  probabilities,
  n_variables = 1L,
  rho = 0,
  structure = c("independent", "exchangeable", "ar1", "custom"),
  correlation = NULL,
  labels = NULL,
  variable_names = NULL,
  seed = NULL,
  batch = NULL
)
```

Arguments

<code>n</code>	Number of observations.
<code>probabilities</code>	A numeric vector used for every variable or a matrix with one row per variable and one column per category.
<code>n_variables</code>	Number of variables when probabilities is a vector.
<code>rho, structure, correlation</code>	Correlation arguments for the latent normal variables. <code>structure</code> follows the same resolution rule as <code>simulate_correlated()</code> : leaving it unset selects "exchangeable" when a non-zero <code>rho</code> is given. Passed to the latent Gaussian generator.
<code>labels</code>	Optional category labels.
<code>variable_names</code>	Optional variable names.
<code>seed</code>	Optional random seed.
<code>batch</code>	Optional single positive whole number. When given, the simulator runs batch times and returns a plain list of batch results, each exactly what the same call without batch returns. With a seed, every dataset gets its own seed drawn from seed, so the whole batch is reproducible; without one the datasets are consecutive draws from the session's random-number stream. The default NULL returns a single result.

Value

A `simulab_sim` base `data.frame` with one row per observation and columns `id` and one column per variable, each holding a category label. Two tidy tables come from `as.data.frame()`: `what = "probabilities"` has one row per variable and category with columns `variable`, `category` and `probability`; `what = "correlation"` has one row per latent correlation-matrix cell with columns `row`, `column` and `correlation`. That correlation is the latent Gaussian one, not the correlation of the realized categories, which is attenuated by the thresholding.

Examples

```
result <- simulate_ordinal(
  n = 200, probabilities = c(0.2, 0.5, 0.3), n_variables = 2, rho = 0.4, seed = 1
```

```
)
head(result)
as.data.frame(result, what = "probabilities")
```

simulate_prediction	<i>Simulate a prediction design with continuous and categorical predictors</i>
---------------------	--

Description

Draws the continuous predictors independently from normal distributions and each categorical predictor by sampling its levels with replacement, then forms the outcome as the intercept plus the continuous linear predictor plus the level effects plus normal residual noise. With no categorical predictor the call is simply [simulate_regression\(\)](#).

Usage

```
simulate_prediction(
  n,
  coefficients,
  categorical_levels = NULL,
  categorical_effects = NULL,
  category_probabilities = NULL,
  predictor_means = 0,
  predictor_sds = 1,
  error_sd = 1,
  outcome = "outcome",
  seed = NULL,
  batch = NULL
)
```

Arguments

n	Sample size. A single whole number of at least 2.
coefficients	Named numeric vector of continuous-predictor coefficients, with an optional (Intercept) entry that defaults to 0 when absent. The remaining names become the continuous predictor columns.
categorical_levels	Named list of levels for categorical predictors, or a tidy data frame with columns variable and level. Optional effect and probability columns in that table supply categorical_effects and category_probabilities, so one table replaces all three arguments. Defaults to NULL, no categorical predictor.
categorical_effects	Named list of level effects matching categorical_levels by name and length, one number per level, added to the outcome for the level a unit is assigned. Required whenever categorical_levels is given.

category_probabilities	Optional named list of sampling probabilities, one non-negative value per level summing to one within each predictor. Defaults to NULL, equal probabilities across a predictor's levels.
predictor_means, predictor_sds	Mean and standard deviation of each normally distributed continuous predictor, a scalar recycled across them (the defaults, 0 and 1) or one value per continuous coefficient.
error_sd	Residual standard deviation. A single positive number, defaulting to 1.
outcome	Outcome-column name. A single non-empty string, defaulting to "outcome".
seed	Optional random seed. A single number, or NULL (the default).
batch	Optional single positive whole number. When given, the simulator runs batch times and returns a plain list of batch results, each exactly what the same call without batch returns. With a seed, every dataset gets its own seed drawn from seed, so the whole batch is reproducible; without one the datasets are consecutive draws from the session's random-number stream. The default NULL returns a single result.

Value

When `categorical_levels` is NULL, whatever `simulate_regression()` returns. Otherwise a `simulab_sim` base data.frame with one row per unit and columns `id`, one column per continuous coefficient name, one character column per categorical predictor, and the outcome named by `outcome`. Components: `coefficients` (one row per fixed term, columns `term` and `coefficient`, always including (Intercept)), `categorical_effects` (one row per predictor level, columns `variable`, `level`, `effect` and `probability`), and `effects` (one row, columns `signal_variance`, `residual_variance` and `r_squared`). `signal_variance` is the *realized* variance of the simulated linear predictor, so `r_squared` is the realized proportion of outcome variance it explains, not a population value.

Examples

```
result <- simulate_prediction(
  n = 200,
  coefficients = c("(Intercept)" = 0.5, x1 = 1, x2 = -1),
  seed = 1
)
head(result)
as.data.frame(result, what = "coefficients")
```

simulate_proportional_survival

Simulate proportional-hazards survival data with calibrated censoring

Description

Draws independent covariates, multiplies the baseline cumulative hazard by $\exp(x \% \% \text{coefficients})$, and inverts it to an event time, so the model is proportional hazards and each coefficient is a log hazard ratio. The three baselines are, for a unit with linear predictor η , Weibull $H(t) = \text{rate} * \exp(\eta) * t^{\text{shape}}$, exponential $H(t) = \text{rate} * \exp(\eta) * t$, and Gompertz $H(t) = \text{rate} * \exp(\eta) * (\exp(\text{shape} * t) - 1) / \text{shape}$. Censoring is random, not administrative: an independent exponential censoring time is drawn for every unit, its rate solved so that the expected censored fraction equals censoring.

Usage

```
simulate_proportional_survival(
  n,
  coefficients,
  baseline = c("weibull", "exponential", "gompertz"),
  rate = 0.1,
  shape = 1,
  censoring = 0.3,
  covariate_distribution = c("normal", "binary"),
  seed = NULL,
  batch = NULL
)
```

Arguments

n	Sample size. A single whole number of at least 5.
coefficients	Covariate log-hazard coefficients, a numeric vector with at least one element and one entry per covariate. Its names become the covariate columns; unnamed vectors are named X1, X2, and so on.
baseline	Baseline hazard family, one of "weibull" (the default), "exponential" or "gompertz".
rate	Positive scale factor of the baseline cumulative hazard. A single positive number, defaulting to 0.1. It is the constant hazard when baseline is "exponential".
shape	Positive shape of the Weibull or Gompertz baseline, in the textbook parameterization of the formulas above (so the hazard rises with time when shape is above 1 for a Weibull baseline). A single positive number, defaulting to 1. It is ignored for an exponential baseline.
censoring	Target fraction of censored observations. A single number in $[0, 1)$, defaulting to 0.3. Exactly 0 turns censoring off and every observation is then an event; the realized fraction is random around the target otherwise.
covariate_distribution	Distribution the covariates are drawn from, independently, one of "normal" (the default, standard normal) or "binary" (Bernoulli with probability 0.5).
seed	Optional random seed. A single number, or NULL (the default).
batch	Optional single positive whole number. When given, the simulator runs batch times and returns a plain list of batch results, each exactly what the same

call without batch returns. With a seed, every dataset gets its own seed drawn from seed, so the whole batch is reproducible; without one the datasets are consecutive draws from the session's random-number stream. The default NULL returns a single result.

Value

A `simulab_sim` base `data.frame` with one row per unit and columns `id`, `time` (the observed time, the smaller of the event and censoring times), the integer status (1 **for an event** and 0 **for a censored observation**), and one covariate column per entry of coefficients. Components: `parameters` (one row per term, columns `term` and `value`, holding the coefficients followed by `baseline_rate`, `shape` and the solved `censoring_rate`) and `diagnostics` (one row, columns `target_censoring`, `realized_censoring` and `baseline`).

Examples

```
result <- simulate_proportional_survival(
  n = 200,
  coefficients = c(x1 = 0.7),
  shape = 1.5,
  censoring = 0.2,
  seed = 1
)
head(result)
as.data.frame(result, what = "parameters")
```

<code>simulate_regression</code>	<i>Simulate a linear regression design</i>
----------------------------------	--

Description

Simulate a linear regression design

Usage

```
simulate_regression(
  n = 100L,
  coefficients = NULL,
  predictor_means = 0,
  predictor_sds = 1,
  correlation = NULL,
  error_sd = 1,
  outcome = "outcome",
  seed = NULL,
  batch = NULL,
  n_predictors = 2L
)
```

Arguments

<code>n</code>	Sample size, defaulting to 100.
<code>coefficients</code>	Named coefficients including optional (Intercept). NULL (the default) draws an intercept from the standard normal distribution and <code>n_predictors</code> slopes, named <code>x1</code> , <code>x2</code> , ..., from the uniform distribution on $[-1, 1]$, each rounded to two decimals, inside the same seeded draw as the data. The drawn values are reported in the coefficients table.
<code>predictor_means</code> , <code>predictor_sds</code>	Predictor means and standard deviations.
<code>correlation</code>	Optional predictor correlation matrix or tidy table.
<code>error_sd</code>	Positive residual standard deviation.
<code>outcome</code>	Name of the outcome variable.
<code>seed</code>	Optional random seed.
<code>batch</code>	Optional single positive whole number. When given, the simulator runs batch times and returns a plain list of batch results, each exactly what the same call without batch returns. With a seed, every dataset gets its own seed drawn from seed, so the whole batch is reproducible; without one the datasets are consecutive draws from the session's random-number stream. The default NULL returns a single result.
<code>n_predictors</code>	Number of predictors when coefficients is NULL, a single positive whole number defaulting to 2. Ignored otherwise.

Value

A `simulab_sim` base data.frame with one row per observation and columns `id`, one column per predictor, and outcome (renamed by `outcome`). Three tidy tables come from `as.data.frame()`: `what = "coefficients"` has one row per term with columns `term` and `coefficient`, the intercept first; `what = "effects"` has a single row with columns `signal_variance` (the quadratic form of the slopes in the predictor covariance matrix), `residual_variance` (`error_sd^2`) and the population `r_squared` `signal_variance / (signal_variance + residual_variance)`; `what = "predictor_correlation"` has one row per correlation-matrix cell with columns `row`, `column` and `correlation`.

Examples

```
# `coefficients` must be named; "(Intercept)" is optional.
result <- simulate_regression(
  n = 200,
  coefficients = c("(Intercept)" = 1, x1 = 0.5, x2 = -0.3),
  seed = 1
)
head(result)
as.data.frame(result, what = "coefficients")
as.data.frame(result, what = "effects")

# With no coefficients, they are drawn and reported as the truth.
drawn <- simulate_regression(seed = 1)
as.data.frame(drawn, what = "coefficients")
```

simulate_scenarios	<i>Run a simulator across a scenario grid</i>
--------------------	---

Description

Every column of scenarios other than `id` and `replication` is passed to the simulator as a named argument, so the column names must be simulator argument names and must not repeat an argument given in `...`. The identifier and replication values are prepended to each generated observation, which makes the result immediately suitable for grouped estimation and recovery checks. Every scenario must produce the same primary columns, otherwise the run is an error.

Usage

```
simulate_scenarios(
  scenarios,
  simulator,
  ...,
  id = "scenario_id",
  replication = "replication",
  seed = NULL
)
```

Arguments

<code>scenarios</code>	A base <code>data.frame</code> with at least one row, commonly from <code>scenario_grid()</code> , containing the <code>id</code> and <code>replication</code> columns.
<code>simulator</code>	Canonical simulator name, taken from the <code>simulator</code> column of <code>list_simulators()</code> and dispatched through <code>simulate_data()</code> .
<code>...</code>	Arguments held constant across scenarios.
<code>id</code>	Single string naming the scenario identifier column of <code>scenarios</code> , default <code>"scenario_id"</code> .
<code>replication</code>	Single string naming the replication column of <code>scenarios</code> , default <code>"replication"</code> .
<code>seed</code>	Optional single base seed. Row <code>i</code> of <code>scenarios</code> is simulated with <code>seed + i - 1</code> , unless <code>scenarios</code> or <code>...</code> already supplies a seed.

Value

A `simulab_sim` base `data.frame` stacking every scenario's output, with one row per generated observation: the `id` and `replication` columns first, then the simulator's own columns. The `scenarios` component holds the grid that was run, one row per scenario.

Examples

```
scenarios <- scenario_grid(mean_b = c(0, 0.5), replications = 2)
result <- simulate_scenarios(
  scenarios, simulator = "ttest",
  n_a = 20, n_b = 20, mean_a = 0, seed = 1
```

```
)
head(result)
```

```
simulate_sequence_batches
```

Simulate repeated sequence datasets

Description

Calls `simulate_sequences()` repetitions times and stacks the results, so each dataset is drawn from its own transition system rather than from a shared one.

Usage

```
simulate_sequence_batches(repetitions = 1L, ..., seed = NULL)
```

Arguments

<code>repetitions</code>	Number of datasets. A single positive whole number, defaulting to 1.
<code>...</code>	Arguments passed to <code>simulate_sequences()</code> .
<code>seed</code>	Optional base seed. A single number, or NULL (the default). Dataset <i>i</i> uses seed + <i>i</i> - 1.

Value

A `simulab_sim` base `data.frame` in long form, one row per dataset, sequence and position, with the column `dataset` followed by the columns `simulate_sequences()` returns (`id`, `period` and `state`). Components: `transitions`, the generating transition probabilities, one row per dataset and from/to pair with columns `dataset`, `from`, `to` and `probability`; and `initial_probabilities`, one row per dataset and state with columns `dataset`, `state` and `probability`.

Examples

```
result <- simulate_sequence_batches(
  repetitions = 3, n = 20, n_states = 3, chain_length = 10, seed = 1
)
head(result)
```

`simulate_sequence_clusters`*Simulate sequences from a mixture of transition systems*

Description

Simulate sequences from a mixture of transition systems

Usage

```
simulate_sequence_clusters(  
  n,  
  transitions,  
  chain_length,  
  proportions = NULL,  
  initial = NULL,  
  labels = NULL,  
  states = NULL,  
  seed = NULL,  
  batch = NULL  
)
```

Arguments

<code>n</code>	Number of sequences.
<code>transitions</code>	List of transition matrices, or a tidy data frame with columns <code>cluster</code> , <code>from</code> , <code>to</code> and <code>probability</code> .
<code>chain_length</code>	Sequence length.
<code>proportions</code>	Cluster proportions. When <code>NULL</code> , clusters are equally likely.
<code>initial</code>	Optional common initial probabilities, shared by every cluster. When <code>NULL</code> , every sequence starts in the first state.
<code>labels</code>	Optional sequence-cluster labels.
<code>states</code>	Optional state labels. When the matrices carry no <code>dimnames</code> and <code>states</code> is <code>NULL</code> , states are labeled by their row index.
<code>seed</code>	Optional random seed.
<code>batch</code>	Optional single positive whole number. When given, the simulator runs <code>batch</code> times and returns a plain list of batch results, each exactly what the same call without <code>batch</code> returns. With a <code>seed</code> , every dataset gets its own seed drawn from <code>seed</code> , so the whole batch is reproducible; without one the datasets are consecutive draws from the session's random-number stream. The default <code>NULL</code> returns a single result.

Value

A long-form `simulab_sim` base data.frame with columns `id`, `period`, `state`, and the true `sequence_cluster`, one row per sequence position. A transitions table (`sequence_cluster`, `from`, `to`, `probability`) is available through `as.data.frame()`.

Examples

```
# One transition matrix per latent cluster.
result <- simulate_sequence_clusters(
  n = 60,
  transitions = list(
    matrix(c(0.8, 0.2, 0.3, 0.7), nrow = 2, byrow = TRUE),
    matrix(c(0.3, 0.7, 0.6, 0.4), nrow = 2, byrow = TRUE)
  ),
  chain_length = 12,
  seed = 1
)
head(result)
components(result)
```

<code>simulate_sequences</code>	<i>Simulate basic or perturbed state sequences</i>
---------------------------------	--

Description

Simulate basic or perturbed state sequences

Usage

```
simulate_sequences(
  n = 100L,
  transition = NULL,
  chain_length = 20L,
  initial = NULL,
  states = NULL,
  n_states = 5L,
  state_categories = c("metacognitive", "cognitive"),
  concentration = 1,
  missing_tail = c(0L, 0L),
  stable_transitions = NULL,
  stability_probability = 0.95,
  instability = c("none", "random_jump", "perturb", "unlikely_jump"),
  instability_probability = 0.4,
  perturbation = 0.5,
  unlikely_threshold = 0.1,
  seed = NULL,
  batch = NULL
)
```

Arguments

<code>n</code>	Number of sequences, defaulting to 100.
<code>transition</code>	Optional transition matrix. When NULL, a random matrix is generated.
<code>chain_length</code>	Maximum sequence length, defaulting to 20.
<code>initial</code>	Initial probabilities or a fixed starting state. When NULL and transition is supplied, every sequence starts in the first state.
<code>states</code>	State labels.
<code>n_states</code>	Number of automatically generated states.
<code>state_categories</code>	Learning-state categories that name a generated state space, defaulting to <code>c("metacognitive", "cognitive")</code> ; see learning_state_categories() . NULL labels the states State 1, State 2, and so on. Ignored when transition or states names the states.
<code>concentration</code>	Positive Dirichlet concentration used for automatic transition and initial probabilities.
<code>missing_tail</code>	Number or range of trailing positions removed from each sequence.
<code>stable_transitions</code>	Optional two-column data frame named from and to defining preferred transitions.
<code>stability_probability</code>	Probability of following a preferred transition.
<code>instability</code>	Instability mechanism for other transitions.
<code>instability_probability</code>	Probability of applying that mechanism.
<code>perturbation</code>	Multiplicative probability perturbation magnitude.
<code>unlikely_threshold</code>	Maximum probability considered unlikely.
<code>seed</code>	Optional random seed.
<code>batch</code>	Optional single positive whole number. When given, the simulator runs batch times and returns a plain list of batch results, each exactly what the same call without batch returns. With a seed, every dataset gets its own seed drawn from seed, so the whole batch is reproducible; without one the datasets are consecutive draws from the session's random-number stream. The default NULL returns a single result.

Value

A long-form `simulab_sim` base `data.frame` with columns `id`, `period`, and `state`, one row per observed sequence position. `as.data.frame(x, what = )` also returns transitions (from, to, probability, row-stochastic in from), `initial_probabilities` (state, probability), `wide` (one row per sequence, columns `id`, `S1`, `S2`, ...), and a one-row settings table.

Examples

```

result <- simulate_sequences(n = 40, n_states = 4, chain_length = 20, seed = 1)
head(result)
components(result)

# With no arguments, a random transition matrix is drawn and reported.
drawn <- simulate_sequences(seed = 1)
as.data.frame(drawn, what = "transitions")

# Preferred transitions followed with a given probability.
stable <- data.frame(
  from = sprintf("State %d", 1:4),
  to = sprintf("State %d", c(2, 3, 4, 1))
)
head(simulate_sequences(
  n = 40, n_states = 4, chain_length = 20, state_categories = NULL,
  stable_transitions = stable, stability_probability = 0.85, seed = 1
))

```

simulate_spline	<i>Add a spline-generated variable</i>
-----------------	--

Description

Add a spline-generated variable

Usage

```

simulate_spline(
  data,
  predictor,
  variable,
  coefficients,
  knots = c(0.25, 0.5, 0.75),
  degree = 3L,
  output_range = NULL,
  noise_variance = 0,
  seed = NULL,
  batch = NULL
)

```

Arguments

data	Base data.frame.
predictor	Predictor variable.
variable	Name of the generated variable.
coefficients	Spline coefficients.

knots	Quantile probabilities for interior knots.
degree	Polynomial degree.
output_range	Optional numeric pair. The spline value v is rescaled to $\text{output_range}[1] + v * \text{diff}(\text{output_range})$, which maps $[0, 1]$ onto the pair. Values outside $[0, 1]$ are rescaled the same way, not clipped.
noise_variance	Non-negative Gaussian noise variance; the noise added to each value has standard deviation $\text{sqrt}(\text{noise_variance})$.
seed	Optional random seed.
batch	Optional single positive whole number. When given, the simulator runs batch times and returns a plain list of batch results, each exactly what the same call without batch returns. With a seed, every dataset gets its own seed drawn from seed, so the whole batch is reproducible; without one the datasets are consecutive draws from the session's random-number stream. The default NULL returns a single result.

Value

A `simulab_sim` base `data.frame` holding data with the spline variable added. `as.data.frame(x, what = "basis")` gives the tidy basis table, one row per observation/basis combination, and `as.data.frame(x, what = "parameters")` gives one row per basis coefficient.

Examples

```
data <- data.frame(id = 1:100, x = stats::runif(100))
result <- simulate_spline(
  data, predictor = "x", variable = "y",
  coefficients = c(0.1, 0.2, 0.5, 0.4, 0.7, 0.6, 0.9),
  knots = c(0.25, 0.5, 0.75), seed = 1
)
head(result)
```

`simulate_study`

Simulate a declaratively specified study

Description

Simulate a declaratively specified study

Usage

```
simulate_study(
  n,
  specification,
  id = "id",
  seed = NULL,
  envir = parent.frame(),
  batch = NULL
)
```

Arguments

n	Number of observational units.
specification	A tidy definition table created with <code>define_variables()</code> , in either the distribution-call form or the formula/variance column form.
id	Name of the identifier column.
seed	Optional random seed. The caller's random-number state is restored on exit.
envir	Environment used to resolve functions and external values named by a formula/variance specification. It is not consulted by the distribution-call form, whose parameters must be expressions over variables defined earlier in the same specification.
batch	Optional single positive whole number. When given, the simulator runs batch times and returns a plain list of batch results, each exactly what the same call without batch returns. With a seed, every dataset gets its own seed drawn from seed, so the whole batch is reproducible; without one the datasets are consecutive draws from the session's random-number stream. The default NULL returns a single result.

Value

A `simulab_sim` base data.frame with n rows: the identifier column followed by one column per variable. Use `as.data.frame(x, what = "definitions")` for the generating definitions.

Examples

```
# `formula` is the mean or linear predictor and may refer to variables
# defined earlier. `variance` is a variance, not a standard deviation, so
# age has sd 10 and outcome has residual sd 2.
simulate_study(
  n = 20,
  specification = define_variables(
    define_variable("age", formula = 40, variance = 100, distribution = "normal"),
    define_variable("treated", formula = 0.5, distribution = "binary"),
    define_variable("outcome", formula = "10 + 2 * treated", variance = 4,
      distribution = "normal")
  ),
  seed = 1
)
```

simulate_survival

Simulate baseline data with survival outcomes

Description

Simulate baseline data with survival outcomes

Usage

```
simulate_survival(
  n,
  specification,
  covariates = NULL,
  id = "id",
  seed = NULL,
  digits = NULL,
  envir = parent.frame(),
  batch = NULL
)
```

Arguments

<code>n</code>	Number of observations.
<code>specification</code>	Survival definitions from define_survivals() , in either the <code>hazard()</code> call form or the column form.
<code>covariates</code>	Optional baseline base data. <code>frame</code> with <code>n</code> rows.
<code>id</code>	Identifier name used when covariates are omitted. Defaults to "id" and is ignored when covariates is supplied.
<code>seed, digits, envir</code>	Arguments passed to augment_survival() .
<code>batch</code>	Optional single positive whole number. When given, the simulator runs batch times and returns a plain list of batch results, each exactly what the same call without batch returns. With a seed, every dataset gets its own seed drawn from seed, so the whole batch is reproducible; without one the datasets are consecutive draws from the session's random-number stream. The default NULL returns a single result.

Value

A `simlab_sim` base data. `frame` with one row per simulated subject: the identifier (or the supplied covariates) followed by one column per event defined in specification. The `survival_definitions` component holds the specification that generated it.

Examples

```
result <- simulate_survival(
  n = 100,
  specification = define_survivals(define_survival("time", formula = -8, shape = 0.3)),
  seed = 1
)
head(result)

# A hazard call states the same process, with the log_rate as an expression.
head(simulate_survival(
  n = 100,
  specification = define_survivals(time = hazard(log_rate = -8, shape = 0.3)),
```

```
    seed = 1
  ))
```

simulate_synthetic	<i>Simulate row-bootstrap synthetic data</i>
--------------------	--

Description

Simulate row-bootstrap synthetic data

Usage

```
simulate_synthetic(
  data,
  n = nrow(data),
  variables = NULL,
  id = "id",
  seed = NULL,
  batch = NULL
)
```

Arguments

data	Source base data.frame.
n	Number of synthetic observations.
variables	Variables to sample. NULL selects every non-identifier variable.
id	Identifier variable in the result.
seed	Optional random seed.
batch	Optional single positive whole number. When given, the simulator runs batch times and returns a plain list of batch results, each exactly what the same call without batch returns. With a seed, every dataset gets its own seed drawn from seed, so the whole batch is reproducible; without one the datasets are consecutive draws from the session's random-number stream. The default NULL returns a single result.

Value

A simulab_sim base data.frame with n rows of jointly resampled values, the identifier column first. as.data.frame(x, what = "provenance") gives the source row each synthetic row was drawn from.

Examples

```
source_data <- data.frame(id = 1:100, x = stats::rnorm(100), y = stats::runif(100))
result <- simulate_synthetic(source_data, n = 50, seed = 1)
head(result)
```

simulate_temporal_network

Simulate a temporal network as edge-activity spells

Description

Simulate a temporal network as edge-activity spells

Usage

```
simulate_temporal_network(
  nodes,
  periods,
  initial_probability = 0.1,
  formation_probability = 0.05,
  dissolution_probability = 0.1,
  directed = TRUE,
  loops = FALSE,
  weight = c("binary", "uniform", "normal", "poisson"),
  weight_mean = 1,
  weight_sd = 1,
  weight_range = c(0.1, 1),
  node_type = NULL,
  seed = NULL,
  batch = NULL
)
```

Arguments

nodes	Number of nodes or unique node labels.
periods	Number of discrete observation periods.
initial_probability	Probability that a dyad is active initially.
formation_probability	Probability that an inactive dyad forms at the next period.
dissolution_probability	Probability that an active dyad dissolves at the next period.
directed	Generate directed dyads.
loops	Permit self-loops.
weight	Edge-spell weight distribution. One weight is drawn per spell and held constant while the spell is active.
weight_mean, weight_sd, weight_range	Weight parameters.
node_type	Optional node type for each node.

seed	Optional seed.
batch	Optional single positive whole number. When given, the simulator runs batch times and returns a plain list of batch results, each exactly what the same call without batch returns. With a seed, every dataset gets its own seed drawn from seed, so the whole batch is reproducible; without one the datasets are consecutive draws from the session's random-number stream. The default NULL returns a single result.

Value

A tidy spell-level `simulab_sim` base data.frame with one row per activity spell and columns from, to, onset, terminus, weight, and censored. `terminus` is exclusive: the spell covers periods onset through `terminus - 1`, so a spell still active in the final period has `terminus == periods + 1` and `censored == TRUE`. `as.data.frame(x, what = )` also returns events (from, to, time, event – "formation" or "dissolution" – and weight, ordered by time), snapshots (one row per active dyad per period: period, from, to, weight), nodes (node, type), and a one-row settings table.

Examples

```
result <- simulate_temporal_network(
  nodes = 20, periods = 4, initial_probability = 0.1, seed = 1
)
head(result)
components(result)
```

simulate_tna	<i>Simulate one transition network</i>
--------------	--

Description

Takes or draws a transition matrix, generates sequences from it with `simulate_sequences()`, and fits one transition network to the sequences with the `tna` package, all in one call.

Usage

```
simulate_tna(
  n = 100L,
  transition = NULL,
  chain_length = 20L,
  initial = NULL,
  states = NULL,
  n_states = 5L,
  state_categories = c("metacognitive", "cognitive"),
  model = c("tna", "ftna", "ctna", "atna"),
  seed = NULL,
  ...,
  batch = NULL
)
```

Arguments

n	Number of sequences, defaulting to 100.
transition	Generating transition matrix, or a tidy data frame with columns from, to and probability. NULL (the default) draws one over n_states states from a Dirichlet distribution.
chain_length	Sequence length, defaulting to 20.
initial	Initial probabilities or a fixed starting state. When NULL and transition is supplied, every sequence starts in the first state.
states	State labels.
n_states	Number of automatically generated states.
state_categories	Learning-state categories the labels of a drawn matrix come from, defaulting to c("metacognitive", "cognitive"); see learning_state_categories() . NULL labels the states State 1, State 2, and so on. Ignored when transition or states names the states.
model	TNA estimator, one of "tna" (the default), "ftna", "ctna" or "atna".
seed	Optional random seed.
...	Further arguments passed to simulate_sequences() , such as concentration or missing_tail.
batch	Optional single positive whole number. When given, the simulator runs batch times and returns a plain list of batch results, each exactly what the same call without batch returns. With a seed, every dataset gets its own seed drawn from seed, so the whole batch is reproducible; without one the datasets are consecutive draws from the session's random-number stream. The default NULL returns a single result.

Value

The fitted network as a native tna model, of class c("simulab_tna", "tna"), so every tna function (plot(), centralities(), summary() and the rest) applies to it. Printing it shows the native tna print, then the generating transition matrix and initial probabilities. Tidy tables come from [as.data.frame\(\)](#): the fitted edges (the default; columns from, to and weight), the generating transitions (columns from, to and probability), initial_probabilities, the generated sequences in long form, wide and model_info. Requires the suggested tna package, and raises simulab_missing_tna without it.

Examples

```
if (requireNamespace("tna", quietly = TRUE)) {
  # With no arguments: a drawn 5-state system, 100 sequences, one network.
  network <- simulate_tna(seed = 1)
  network
  validate_recovery(network, network, term = "to", estimate = "weight")

  # From a given transition matrix.
  transition <- data.frame(
```

```

    from = rep(c("plan", "act"), each = 2),
    to = rep(c("plan", "act"), times = 2),
    probability = c(0.3, 0.7, 0.6, 0.4)
  )
  simulate_tna(n = 200, transition = transition, seed = 1)
}

```

simulate_tna_batches *Simulate and fit repeated TNA networks*

Description

Generates repetitions sequence datasets with `simulate_sequence_batches()` and fits one transition network to each with `fit_tna()`, so the estimated networks can be held against the transition systems that generated them.

Usage

```

simulate_tna_batches(
  repetitions = 1L,
  model = c("tna", "ftna", "ctna", "atna"),
  ...,
  seed = NULL
)

```

Arguments

<code>repetitions</code>	Number of fitted networks. A single positive whole number, defaulting to 1, so a call with no arguments draws one transition system, generates sequences from it and fits one network.
<code>model</code>	TNA estimator, one of "tna" (the default), "ftna", "ctna" or "atna".
<code>...</code>	Arguments passed to <code>simulate_sequences()</code> .
<code>seed</code>	Optional base seed. A single number, or NULL (the default). Dataset <i>i</i> uses seed + <i>i</i> - 1.

Value

A `simulab_sim` base data.frame of fitted edges, one row per dataset and from/to pair, with columns `dataset`, `from`, `to` and `weight`. Printing it shows `true_transitions` as the truth, keyed by the same dataset. Components: `sequences`, the generated sequences in long form with columns `dataset`, `id`, `period` and `state`; `true_transitions`, the generating probabilities with columns `dataset`, `from`, `to` and `probability`; and `model_info`, one row per dataset describing the fit.

Examples

```

if (requireNamespace("tna", quietly = TRUE)) {
  head(simulate_tna_batches(
    repetitions = 2, model = "tna", n = 20, n_states = 3, chain_length = 10, seed = 1
  ))

  # One network with no arguments, recovered against its own truth.
  network <- simulate_tna_batches(seed = 1)
  head(validate_recovery(network, network, term = "to", estimate = "weight"))
}

```

simulate_tna_network	<i>Simulate a grouped TNA transition network</i>
----------------------	--

Description

Simulate a grouped TNA transition network

Usage

```

simulate_tna_network(
  groups,
  nodes_per_group,
  group_names = NULL,
  within_probability = 0.4,
  between_probability = 0.15,
  loops = FALSE,
  seed = NULL,
  batch = NULL
)

```

Arguments

groups	Number of node groups.
nodes_per_group	Nodes per group, scalar or vector.
group_names	Optional group names.
within_probability, between_probability	Edge probabilities within and between node groups.
loops	Permit self transitions when drawing the underlying edges. A node left with no outgoing edge is nevertheless given a self transition of probability one, so self-loops can appear even when loops is FALSE.
seed	Optional random seed.

batch Optional single positive whole number. When given, the simulator runs batch times and returns a plain list of batch results, each exactly what the same call without batch returns. With a seed, every dataset gets its own seed drawn from seed, so the whole batch is reproducible; without one the datasets are consecutive draws from the session's random-number stream. The default NULL returns a single result.

Value

A tidy `simulab_sim` base `data.frame` holding the non-zero transitions only, one row per edge, with columns `from`, `to`, and `probability`. `as.data.frame(x, what = )` also returns `nodes` (node, group), `adjacency` (the long row/column/weight table of the underlying directed Bernoulli graph), and `transitions` (the full nodes^2 -row transition table including zeros). The transition matrix is row-stochastic: every from node's probabilities sum to one.

Examples

```
result <- simulate_tna_network(groups = 3, nodes_per_group = 4, seed = 1)
head(result)
components(result)
```

simulate_ttest

Simulate a two-group design

Description

Simulate a two-group design

Usage

```
simulate_ttest(
  n_a = 50L,
  n_b = 50L,
  mean_a = NULL,
  mean_b = NULL,
  sd_a = 1,
  sd_b = 1,
  labels = c("A", "B"),
  outcome = "outcome",
  seed = NULL,
  batch = NULL
)
```

Arguments

`n_a`, `n_b` Group sample sizes, each defaulting to 50.

mean_a, mean_b	Group means. A mean left NULL (the default) is drawn from the standard normal distribution and rounded to two decimals, inside the same seeded draw as the data, and is reported in the parameters table.
sd_a, sd_b	Positive group standard deviations.
labels	Two group labels.
outcome	Name of the outcome variable.
seed	Optional random seed.
batch	Optional single positive whole number. When given, the simulator runs batch times and returns a plain list of batch results, each exactly what the same call without batch returns. With a seed, every dataset gets its own seed drawn from seed, so the whole batch is reproducible; without one the datasets are consecutive draws from the session's random-number stream. The default NULL returns a single result.

Value

A `simulab_sim` base `data.frame` with one row per observation and columns `id`, `group` and `outcome` (renamed by `outcome`). Two tidy tables come from `as.data.frame()`: `what = "parameters"` has one row per group with columns `group`, `n`, `mean` and `sd`; `what = "effects"` has a single row with columns `contrast`, `mean_difference`, `pooled_sd` and `cohens_d`, the population Cohen's *d* formed from the sample-size-weighted pooled standard deviation.

Examples

```
result <- simulate_ttest(n_a = 40, n_b = 40, mean_a = 0, mean_b = 0.6, seed = 1)
head(result)
as.data.frame(result, what = "parameters")

# With no arguments, the group means are drawn and reported.
drawn <- simulate_ttest(seed = 1)
as.data.frame(drawn, what = "parameters")
```

simulate_until_event *Simulate events and retain records through the nth event*

Description

Simulate events and retain records through the *nth* event

Usage

```
simulate_until_event(
  data,
  definition,
  occurrence = 1L,
  id = "id",
```

```

    period = "period",
    seed = NULL,
    envir = parent.frame(),
    batch = NULL
  )

```

Arguments

<code>data</code>	Long-form base data.frame.
<code>definition</code>	One binary variable definition from <code>define_variable()</code> .
<code>occurrence, id, period</code>	Event trimming arguments.
<code>seed</code>	Optional random seed.
<code>envir</code>	Formula evaluation environment.
<code>batch</code>	Optional single positive whole number. When given, the simulator runs batch times and returns a plain list of batch results, each exactly what the same call without batch returns. With a seed, every dataset gets its own seed drawn from seed, so the whole batch is reproducible; without one the datasets are consecutive draws from the session's random-number stream. The default NULL returns a single result.

Value

A `simulab_sim` base data.frame with the generated event indicator and records through the requested occurrence.

Examples

```

# Long-form input: one row per unit per period.
data <- expand_periods(data.frame(id = 1:30), periods = 8,
                      id = "id", period = "period")

result <- simulate_until_event(
  data,
  definition = define_variable("event", formula = "0.3", distribution = "binary"),
  occurrence = 1,
  seed = 1
)
head(result)

```

simulation_scenarios *List built-in simulation scenarios*

Description

List built-in simulation scenarios

Usage

```
simulation_scenarios()
```

Value

A base data.frame with one row per scenario and columns scenario, family and description. Every scenario is accepted by [run_simulation_scenario\(\)](#).

Examples

```
simulation_scenarios()
```

spline_basis	<i>Compute tidy B-spline basis functions</i>
--------------	--

Description

Compute tidy B-spline basis functions

Usage

```
spline_basis(knots = c(0.25, 0.5, 0.75), degree = 3L, n = 1000L)
```

Arguments

knots	Interior knots between zero and one.
degree	Polynomial degree.
n	Number of evaluation points.

Value

A base data.frame with one row per x/basis combination and columns x, basis, and value.

Examples

```
head(spline_basis(knots = c(0.25, 0.5, 0.75), degree = 3, n = 20))
```

spline_curves	<i>Compute one or more tidy spline curves</i>
---------------	---

Description

Compute one or more tidy spline curves

Usage

```
spline_curves(coefficients, knots = c(0.25, 0.5, 0.75), degree = 3L, n = 1000L)
```

Arguments

`coefficients` Numeric vector or matrix with one coefficient column per curve.
`knots, degree, n` Spline-basis arguments.

Value

A base `data.frame` with one row per `x`/curve combination and columns `x`, `curve`, and `value`.

Examples

```
head(spline_curves(coefficients = c(0.1, 0.2, 0.5, 0.4, 0.7, 0.6, 0.9),
  knots = c(0.25, 0.5, 0.75), degree = 3, n = 20))
```

summarize_networks	<i>Summarize multiple networks</i>
--------------------	------------------------------------

Description

Summarize multiple networks

Usage

```
summarize_networks(networks, threshold = 0, directed = TRUE)
```

Arguments

<code>networks</code>	Named list of supported networks. Unnamed elements are labeled Network 1, Network 2, and so on.
<code>threshold</code>	Edge-presence threshold, applied as <code>abs(weight) > threshold</code> . It affects the edges and <code>mean_weight</code> columns only; density and components are computed from every edge in the network, thresholded or not.
<code>directed</code>	Treat networks as directed. density is the directed edge density when TRUE; components always counts weakly connected components.

Value

A tidy base data.frame with one row per network and columns network, nodes, edges, density, mean_weight, and components. mean_weight is NA when no edge passes the threshold. Requires the suggested igraph package.

Examples

```
networks <- list(
  first = simulate_network(nodes = 25, model = "bernoulli", probability = 0.1, seed = 1),
  second = simulate_network(nodes = 25, model = "bernoulli", probability = 0.2, seed = 2)
)
if (requireNamespace("igraph", quietly = TRUE)) {
  summarize_networks(networks)
}
```

summarize_simulations *Summarize simulated numeric variables*

Description

Summarize simulated numeric variables

Usage

```
summarize_simulations(data, by = NULL, variables = NULL)
```

Arguments

data	Simulation data: a simulab_sim or a plain base data.frame.
by	Optional character vector of grouping column names. NULL, the default, summarizes all rows together.
variables	Character vector of numeric variables to summarize. NULL, the default, selects every numeric non-grouping column.

Value

A plain base data.frame with one row per group and variable: the by columns (none when by is NULL), then variable, observations (the non-missing count), mean, sd, minimum and maximum. Missing values are left out of the statistics; a group with none left reports NA statistics.

Examples

```
data <- simulate_ttest(n_a = 50, n_b = 50, mean_a = 0, mean_b = 0.6, seed = 1)
summarize_simulations(data, by = "group", variables = "outcome")
```

`summarize_transitions` *Count observed transitions in sequence data*

Description

Count observed transitions in sequence data

Usage

```
summarize_transitions(  
  data,  
  id = "id",  
  period = "period",  
  state = "state",  
  normalize = TRUE  
)
```

Arguments

<code>data</code>	Long-form sequence data.
<code>id, period, state</code>	Column names.
<code>normalize</code>	Return row-conditional transition proportions. When FALSE, the probability column is kept but filled with NA.

Value

A tidy base `data.frame` with one row per observed from/to pair and columns `from`, `to`, `count`, and `probability`. Pairs never observed are absent rather than zero, so probability sums to one within each from state when `normalize` is TRUE.

Examples

```
data <- simulate_markov(  
  n = 40,  
  transition = matrix(c(0.7, 0.3, 0.4, 0.6), nrow = 2, byrow = TRUE),  
  chain_length = 20, states = c("A", "B"), seed = 1  
)  
summarize_transitions(data, normalize = TRUE)
```

summary.simulab_sim	<i>Summarize simulated variables</i>
---------------------	--------------------------------------

Description

Summarizes the primary data only; the other components are not included.

Usage

```
## S3 method for class 'simulab_sim'
summary(object, ...)
```

Arguments

object	A simulab_sim object.
...	Reserved for future methods.

Value

A base data.frame with one row per column of the primary data and the columns variable, class, observations, missing, unique, mean, sd, minimum and maximum. The last four are NA for non-numeric columns.

Examples

```
result <- simulate_ttest(n_a = 30, n_b = 30, mean_a = 0, mean_b = 0.5, seed = 1)
summary(result)
```

survival_curve	<i>Compute a tidy Weibull survival curve</i>
----------------	--

Description

Evaluates the simstudy Weibull curve of [define_survival\(\)](#), $S(t) = \exp(-\exp(\text{formula}) * t^{(1 / \text{shape})} / \text{scale}))$, at n survival probabilities spread evenly from $1 - 1 / n$ down to $1 / n$, inverting it for the matching times. The points are therefore equally spaced in survival probability, not in time.

Usage

```
survival_curve(formula, shape, scale = 1, n = 100L, time_limits = NULL)
```

Arguments

formula	Intercept of the linear predictor on the log scale. A single finite number. <code>exp(formula)</code> multiplies the cumulative hazard, and equals the hazard only when shape and scale are both 1.
shape	Positive Weibull shape in the <code>simstudy</code> parameterization, the exponent applied to the transformed time. A single positive number.
scale	Positive Weibull scale, dividing the cumulative hazard. A single positive number, defaulting to 1.
n	Number of curve points. A single whole number of at least 2, defaulting to 100.
time_limits	Optional inclusive time range, as an increasing non-negative numeric vector of length 2, that the curve is restricted to. Defaults to <code>NULL</code> , the whole curve. Points outside the range are dropped, so fewer than <code>n</code> rows are returned.

Value

A `base.data.frame` with one row per retained curve point, in increasing time order, and columns `time` and `survival`.

Examples

```
curve <- survival_curve(formula = -8, shape = 0.3, n = 10)
head(curve)
```

trim_events	<i>Trim longitudinal records at an event occurrence</i>
-------------	---

Description

Trim longitudinal records at an event occurrence

Usage

```
trim_events(data, id, period, event, occurrence = 1L, event_value = 1)
```

Arguments

data	Long-form base <code>data.frame</code> .
id	Unit identifier.
period	Ordered period variable.
event	Event indicator variable.
occurrence	Event occurrence at which to stop.
event_value	Value identifying an event.

Value

A `simulab_sim` base data.frame retaining observations through the requested event, or all observations when it never occurs.

Examples

```
# A long-form event log: `simulate_until_event()` generates the indicator,
# `trim_events()` then keeps records through its first occurrence.
data <- simulate_until_event(
  expand_periods(data.frame(id = 1:20), periods = 10,
    id = "id", period = "period"),
  definition = define_variable("event", formula = "0.25",
    distribution = "binary"),
  seed = 1
)

result <- trim_events(
  data, id = "id", period = "period", event = "event", occurrence = 1
)
head(result)
```

update_definition	<i>Update one variable definition</i>
-------------------	---------------------------------------

Description

Update one variable definition

Usage

```
update_definition(
  specification,
  variable,
  formula = NULL,
  variance = NULL,
  distribution = NULL,
  link = NULL
)
```

Arguments

specification A `simulab_spec` object in the formula/variance column form, as created by `define_variable()`, `repeat_variables()` or `read_definitions()`.

variable Variable to update.

formula, variance, distribution, link Replacement values. NULL keeps the existing value.

Value

An updated `simulab_spec` base data.frame.

Examples

```
specification <- define_variables(
  define_variable("a", formula = "0", variance = "1", distribution = "normal")
)
update_definition(specification, variable = "a", formula = "5")
```

`validate_recovery`
Compare recovered estimates with known simulation truth

Description

Compare recovered estimates with known simulation truth

Usage

```
validate_recovery(
  estimates,
  truth,
  term = "term",
  estimate = "estimate",
  true_value = NULL,
  tolerance = 0.1
)
```

Arguments

<code>estimates</code>	Tidy base data.frame with a term column and an estimate column, or a fitted model with named coefficients, such as the result of <code>lm()</code> , whose coefficients become the term and estimate columns.
<code>truth</code>	The generating values. A tidy base data.frame with a term column and a true-value column; a <code>simulab_sim</code> result, whose table of generating values (the one its print method shows under <code>Truth</code>) is used; or a batch of results from <code>batch = n</code> , whose tables are stacked with a <code>batch_id</code> column numbered as <code>apply_batch()</code> numbers the data sets, so each estimate is compared with the truth of its own data set.
<code>term</code>	Name of the term column in both frames, default <code>"term"</code> . The frames are merged on it and on every other column the two share, such as <code>batch_id</code> .
<code>estimate</code>	Name of the estimate column in estimates, default <code>"estimate"</code> .
<code>true_value</code>	Name of the true-value column in truth. <code>NULL</code> (the default) uses a column named <code>truth</code> when there is one, and otherwise the only numeric column of truth other than the shared identifiers; it is an error when there are several.
<code>tolerance</code>	Single non-negative absolute-error tolerance for successful recovery, default <code>0.1</code> .

Value

A base data.frame with one row per row of estimates, in the same order, plus one row for each term found only in truth. Every column of estimates other than term and estimate (for example the batch_id of `apply_batch()`) comes first, then the columns term, estimate, truth, bias (estimate minus truth), absolute_error, relative_error (NA where truth is zero) and recovered, a logical that is TRUE when absolute_error is at most tolerance.

Examples

```
estimates <- data.frame(term = c("x1", "x2"), estimate = c(0.52, -0.28))
truth <- data.frame(term = c("x1", "x2"), truth = c(0.5, -0.3))
validate_recovery(estimates, truth, tolerance = 0.1)

# One data set: the fitted model against the result's own truth.
data_set <- simulate_regression(seed = 1)
validate_recovery(lm(outcome ~ x1 + x2, data = data_set), data_set)

# A batch with drawn coefficients: each data set is compared with its own.
datasets <- simulate_regression(seed = 1, batch = 20)
fit_coefficients <- function(data) {
  fit <- lm(outcome ~ x1 + x2, data = data)
  data.frame(term = names(coef(fit)), estimate = unname(coef(fit)))
}
estimates <- apply_batch(datasets, fit_coefficients)
head(validate_recovery(estimates, datasets))
```

write_simulation	<i>Export a simulation component</i>
------------------	--------------------------------------

Description

Export a simulation component

Usage

```
write_simulation(x, file, what = "data")
```

Arguments

x	A simulab_sim simulation result.
file	Destination path. The extension decides the format and must be .csv (written with <code>utils::write.csv()</code> , without row names) or .rds; any other extension is an error.
what	Single string naming the component to write, default "data". Use <code>components(x)</code> to list the choices.

Value

The normalized output path, invisibly.

Examples

```
result <- simulate_ttest(n_a = 20, n_b = 20, mean_a = 0, mean_b = 0.5, seed = 1)
file <- tempfile(fileext = ".csv")
write_simulation(result, file = file)
head(utils::read.csv(file))
unlink(file)
```

Index

`apply_batch`, 5
`apply_batch()`, 144, 145
`apply_conditions`, 6
`as.data.frame()`, 131
`as.data.frame.simulab_sim`, 7
`as.data.frame.simulab_tna`, 8
`as_igraph`, 9
`as_igraph()`, 63
`as_tna_model`, 9
`assess_tna_reliability`, 10
`assign_stepped_wedge`, 11
`assign_treatment`, 13
`augment_correlated`, 14
`augment_density`, 15
`augment_factorial`, 16
`augment_markov`, 17
`augment_study`, 18
`augment_study()`, 43
`augment_survival`, 19
`augment_survival()`, 127
`augment_synthetic`, 20

`block_correlation`, 21
`bootstrap_tna`, 22

`calibrate_distribution`, 23
`calibrate_icc`, 24
`calibrate_logistic`, 25
`calibrate_moments`, 26
`calibrate_moments()`, 23
`calibrate_survival`, 27
`categorical_formula`, 28
`combine_competing_risks`, 29
`combine_competing_risks()`, 19
`compare_centralities`, 30
`compare_networks`, 31
`compare_networks()`, 11, 34, 35, 47
`compare_tna_models`, 32
`components`, 33
`correlation_structure`, 33

`cross_validate_tna`, 34

`define_condition`, 35
`define_condition()`, 36
`define_conditions`, 36
`define_conditions()`, 6
`define_missingness`, 37
`define_missingness()`, 38
`define_missingnesses`, 38
`define_missingnesses()`, 61
`define_survival`, 39
`define_survival()`, 19, 27, 40, 141
`define_survivals`, 40
`define_survivals()`, 19, 127
`define_variable`, 41
`define_variable()`, 14, 43, 143
`define_variables`, 42
`define_variables()`, 18, 78, 126

`encode_factors`, 44
`encode_sequences`, 45
`evaluate_edge_recovery`, 46
`evaluate_edge_recovery()`, 47
`evaluate_tna_estimation`, 46
`expand_clusters`, 48
`expand_periods`, 49
`expand_periods()`, 11

`factorial_design`, 50
`factorial_design()`, 16
`fit_tna`, 51
`fit_tna()`, 11, 22, 23, 35, 47, 52, 53, 72, 132
`fit_tna_batch`, 52

`generate_transition_system`, 53
`generate_transition_system()`, 47
`global_name_regions`, 54
`global_name_regions()`, 55
`global_names`, 55

`inject_missingness`, 55

- learning_state_categories, [57](#)
- learning_state_categories(), [54](#), [57](#), [123](#), [131](#)
- learning_states, [57](#)
- learning_states(), [71](#)
- linear_formula, [58](#)
- list_distributions, [58](#)
- list_distributions(), [27](#), [36](#), [42](#)
- list_simulators, [59](#)
- list_simulators(), [82](#), [119](#)
- merge_studies, [60](#)
- missingness_matrix, [60](#)
- missingness_matrix(), [38](#), [65](#)
- mixture_formula, [62](#)
- network centrality, [62](#)
- observe_treatment, [63](#)
- observed_data, [64](#)
- observed_data(), [60](#)
- parameter_grid, [65](#)
- print.simulab_sim, [66](#)
- print.simulab_tna, [67](#)
- read_definitions, [68](#)
- read_definitions(), [143](#)
- repeat_variables, [68](#)
- repeat_variables(), [43](#), [143](#)
- run_simulation_scenario, [69](#)
- run_simulation_scenario(), [137](#)
- sample_global_names, [70](#)
- sample_learning_states, [71](#)
- sample_tna, [71](#)
- scenario_grid, [73](#)
- scenario_grid(), [119](#)
- select_variables, [73](#)
- simulate_anova, [74](#)
- simulate_bipartite_network, [75](#)
- simulate_clusters, [76](#)
- simulate_copula, [78](#)
- simulate_copula(), [14](#), [58](#)
- simulate_correlated, [80](#)
- simulate_correlated(), [81](#), [82](#), [113](#)
- simulate_correlation, [81](#)
- simulate_data, [82](#)
- simulate_data(), [59](#), [119](#)
- simulate_density, [83](#)
- simulate_density(), [15](#)
- simulate_edge_list, [84](#)
- simulate_event_log, [85](#)
- simulate_factors, [87](#)
- simulate_group_sequences, [88](#)
- simulate_group_tna, [90](#)
- simulate_growth, [91](#)
- simulate_hmm, [93](#)
- simulate_irt, [94](#)
- simulate_lca, [96](#)
- simulate_longitudinal, [98](#)
- simulate_lpa, [100](#)
- simulate_markov, [101](#)
- simulate_ml_lpa, [103](#)
- simulate_multilevel, [105](#)
- simulate_multiplex_network, [107](#)
- simulate_network, [108](#)
- simulate_network(), [110](#), [111](#)
- simulate_network_batches, [110](#)
- simulate_network_matrix, [111](#)
- simulate_ordinal, [112](#)
- simulate_prediction, [114](#)
- simulate_proportional_survival, [115](#)
- simulate_regression, [117](#)
- simulate_regression(), [114](#), [115](#)
- simulate_scenarios, [119](#)
- simulate_sequence_batches, [120](#)
- simulate_sequence_batches(), [132](#)
- simulate_sequence_clusters, [121](#)
- simulate_sequences, [122](#)
- simulate_sequences(), [47](#), [120](#), [130–132](#)
- simulate_spline, [124](#)
- simulate_study, [125](#)
- simulate_study(), [43](#)
- simulate_survival, [126](#)
- simulate_synthetic, [128](#)
- simulate_temporal_network, [129](#)
- simulate_tna, [130](#)
- simulate_tna(), [8](#), [67](#)
- simulate_tna_batches, [132](#)
- simulate_tna_network, [133](#)
- simulate_ttest, [134](#)
- simulate_until_event, [135](#)
- simulation_scenarios, [136](#)
- simulation_scenarios(), [69](#)
- spline_basis, [137](#)
- spline_curves, [138](#)
- summarize_networks, [138](#)

`summarize_simulations`, [139](#)
`summarize_transitions`, [140](#)
`summary.simulab_sim`, [141](#)
`survival_curve`, [141](#)
`survival_curve()`, [27](#)

`trim_events`, [142](#)

`update_definition`, [143](#)

`validate_recovery`, [144](#)

`write_simulation`, [145](#)