# Package 'flexsynth'

September 2, 2026

**Type** Package

**Title** Flexible Synthetic Data for Nested, Longitudinal and Linked Multi-Table Data

**Version** 0.2.1

**Description** Generates utility-oriented synthetic data for supported flat, nested, longitudinal and tree-linked multi-table designs, including patients, admissions, procedures and laboratory results linked by identifiers. The default engine uses sequential conditional synthesis; an opt-in differentially private engine implements person-level (epsilon, delta) mechanisms and records their budget accounting. Synthetic output is not anonymisation. Empirical utility and disclosure-risk diagnostics are descriptive and do not by themselves establish that a release is safe.

**License** MIT + file LICENSE

**Encoding** UTF-8

**RoxygenNote** 7.3.3

**Depends** R (>= 4.1)

**Imports** graphics, grDevices, parallel, stats, utils

**Suggests** testthat (>= 3.0.0), rpart, ranger, partykit, data.table, knitr, litedown

**Config/testthat/edition** 3

**VignetteBuilder** litedown

**URL** https://github.com/lauyeehow1986-hub/Flexsynth

**BugReports** https://github.com/lauyeehow1986-hub/Flexsynth/issues

**NeedsCompilation** no

**Author** Yee How Lau [aut, cre]

**Maintainer** Yee How Lau <lauyeehow1986@gmail.com>

**Repository** CRAN

**Date/Publication** 2026-09-02 12:00:02 UTC

# Contents



---

`as.data.frame.synth_result` *Extract the synthetic data frame from a synth_result*

---

## Description

Extract the synthetic data frame from a synth_result

## Usage

```
## S3 method for class 'synth_result'
as.data.frame(x, ...)
```

## Arguments

| | |
|---|---|
| `x` | A `synth_result`. |
| `...` | Unused. |

## Value

The synthetic `data.frame` (for `m == 1`).

## as.list.synth_linked_result *Extract the synthetic tables from a synth_linked_result*

### Description

Extract the synthetic tables from a synth_linked_result

### Usage

```
## S3 method for class 'synth_linked_result'
as.list(x, ...)
```

### Arguments

| | |
|---|---|
| x | A synth_linked_result. |
| ... | Unused. |

### Value

A named list of synthetic data.frames (for m == 1).

## check_linkage *Check referential integrity of linked tables*

### Description

Verify that a set of linked tables is internally consistent: every table's key identifies its rows uniquely, and every child row's foreign key resolves to an existing parent row (no orphans). Works on a synth_linked() result or on a raw named list of tables (supply keys in that case).

### Usage

```
check_linkage(object, keys = NULL, verbose = TRUE)
```

### Arguments

| | |
|---|---|
| object | A synth_linked_result, or a named list of data.frames. |
| keys | When object is a raw list of tables, a named list of key columns (as passed to synth_linked()). Ignored for a result object. |
| verbose | Logical; print a short summary. Defaults to TRUE. |

### Value

Invisibly, a report data.frame (one row per table, or per table and collection when m > 1) with an ok attribute that is TRUE when there are no duplicate keys and no orphan rows.

## Examples

```
tabs <- list(
  patients   = data.frame(id = 1:3),
  admissions = data.frame(id = c(1, 2, 9), admission_id = c(1, 1, 1))
)
check_linkage(tabs, keys = list(patients = "id",
                                admissions = c("id", "admission_id")))
```

---

`compare_estimates` *Specific-utility comparison of an analysis on real vs synthetic data*

---

## Description

Fit the same analysis on the real data and on the synthetic data and compare the estimates. This is the *analysis-specific* utility measure — "does my regression come out the same?" — complementing the general marginal / pMSE diagnostics of `diagnose()`. For each term it reports the confidence-interval overlap (Karr et al. 2006; 1 = identical intervals, 0 = just touching, negative = disjoint) and the standardised difference of the estimates on the real standard-error scale (smaller is better).

## Usage

```
compare_estimates(
  real,
  syn,
  analysis,
  rule = c("synthpop", "reiter"),
  population_inference = TRUE,
  conf.level = 0.95,
  ...
)
```

## Arguments

| | |
|---|---|
| `real` | The real `data.frame`. |
| `syn` | A `synth_result` (pooled) or a synthetic `data.frame`. |
| `analysis` | A function of one `data.frame` returning a fitted model (`coef()`/`vcov()`) or a `list(estimate =, variance =)`. |
| `rule`, `population_inference` | Passed to `pool_synth()` when `syn` is a multi-dataset result. |
| `conf.level` | Confidence level for both intervals (default `0.95`). |
| `...` | Reserved. |

## Details

When `syn` is a multi-dataset `synth()` result its estimates are pooled with `pool_synth()` (so the synthetic standard errors reflect synthesis); a single synthetic `data.frame` is analysed directly. `analysis` uses the same contract as `pool_synth()`.

## Value

A `flexsynth_utility` object; `$estimates` has `term`, `est_real`, `est_syn`, `overlap`, `std_diff`, and both intervals.

## See Also

`pool_synth()`, `diagnose()`.

## Examples

```
d <- data.frame(id = 1:500, x = rnorm(500))
d$y <- 1 + 2 * d$x + rnorm(500)
res <- synth(d, ~ id, m = 5, seed = 1)
compare_estimates(d, res, function(dat) lm(y ~ x, dat))
```

---

# diagnose *Utility diagnostics comparing synthetic data to the real data*

---

## Description

Quantifies how faithfully synthetic data reproduces the real data's distributions and dependence structure. Because Track A output carries no formal privacy guarantee, these utility diagnostics (and the risk diagnostics in `disclosure_risk()`) are how synthesis quality is judged.

## Usage

```
diagnose(real, syn, vars = NULL, propensity = "logistic", ...)
```

## Arguments

| | |
|---|---|
| `real` | The real `data.frame` (or a named list of them for linked data). |
| `syn` | The synthetic data: a `data.frame`, a `synth()` `synth_result`, a named list of tables, or a `synth_linked()` `synth_linked_result`. |
| `vars` | Optional character vector restricting the variables compared; defaults to all columns present in both. |
| `propensity` | Propensity model for the pMSE, `"logistic"` (default) or `"cart"` (needs the `rpart` package). |
| `...` | Unused. |

## Details

Four views are reported:

- **Univariate** — per-variable marginal fit: a Kolmogorov-Smirnov statistic for numeric variables and a total-variation distance for categorical ones (both in $[0, 1]$; smaller is better).
- **Correlation** — the Frobenius and mean absolute difference between the real and synthetic numeric correlation matrices.

- **Association** — the mean and maximum absolute difference between the real and synthetic categorical association (Cramer's V) matrices, so dependence between factor variables is checked, not just numeric ones.
- **Propensity (pMSE)** — a descriptive utility score: a model is fitted to tell real from synthetic records; when they are indistinguishable the propensity mean squared error sits near its null expectation, so the logistic model's `ratio` is near 1 (larger means more distinguishable). This is an in-sample, row-level diagnostic: the default logistic model has main effects only, and repeated rows in longitudinal data are not treated as independent inferential evidence. CART does not have an analytic null ratio. Treat all results as descriptive rather than hypothesis tests.

If `real` and `syn` are named lists (or `syn` is a `synth_linked_result`), each table is diagnosed and a per-table result is returned.

## Value

A `flexsynth_diagnostics` object (or a `flexsynth_diagnostics_list` for linked / list input). Has `print()` and `plot()` methods.

## See Also

`disclosure_risk()` for privacy risk.

## Examples

```
df <- data.frame(
  id  = 1:200,
  age = round(rnorm(200, 60, 10)),
  sbp = round(rnorm(200, 130, 15))
)
res <- synth(df, ~ id, seed = 1)
diagnose(df, res)
```

---

# disclosure_risk *Empirical disclosure-risk diagnostics for synthetic data*

---

## Description

Reports how much a synthetic dataset could leak about the real records it was trained on. Four complementary measures are computed:

## Usage

```
disclosure_risk(
  real,
  syn,
  quasi = NULL,
  target = NULL,
  holdout = NULL,
```

```
  max_records = 2000L,
  seed = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| real | The real data.frame (or named list) the data was synthesised from. |
| syn | The synthetic data: a data.frame, synth() synth_result, named list, or synth_linked() synth_linked_result. |
| quasi | Character vector of quasi-identifier columns; defaults to all columns present in both frames. |
| target | Optional single sensitive column for the attribute-disclosure (TCAP) measure; the quasi-identifiers minus target are used as the keys. NULL (default) skips it. For a linked (list) input it is assessed only on the table that contains it. |
| holdout | Optional data.frame of real records *excluded* from synthesis, enabling the membership-inference check. For linked input, a named list containing a holdout data.frame for every evaluated table. |
| max_records | Cap on the number of rows used for the distance computations (each of real / synthetic is sampled down to this); keeps the pairwise distances tractable. Default 2000. |
| seed | Optional integer seed for the subsampling. |
| ... | Unused. |

## Details

- **Replicated uniques** — records that are unique in the real data on the quasi-identifiers and are nonetheless reproduced exactly in the synthetic data. These are the classic identity-disclosure risks.
- **Distance to closest record (DCR)** — for each synthetic record, the Gower distance to the nearest real record. A DCR of 0 is an exact copy. The synthetic-to-real distances are compared against the real-to-real nearest-neighbour distances: if synthetic records are not systematically closer to real records than real records are to each other, identity risk is low.
- **Membership inference** — if a holdout of records *not* used for synthesis is supplied, an attacker who guesses "member" for records close to the synthetic data is simulated. The reported AUC (0.5 = no advantage) and advantage (2 * AUC - 1) measure how well training membership can be inferred.
- **Attribute disclosure (TCAP)** — if a sensitive target column is named, the Target Correct Attribution Probability (Taub, Elliot et al.): an attacker who knows a real record's quasi-identifier keys reads the conditional distribution of the target off the synthetic records that share those keys. tcap is the mean synthetic probability of the true target over matched records, baseline the marginal-only attacker evaluated on those same covered records, and lift the excess the key-conditioning buys. baseline_unconditional is also returned for context. Meant for categorical keys / target.

Quasi-identifiers should be the genuinely identifying columns; exclude surrogate keys such as a regenerated unit id (a synthetic id never matches a real one, which would understate risk). If real and syn are named lists (or syn is a synth_linked_result), each table is assessed separately.

## Value

A `flexsynth_disclosure` object (or `flexsynth_disclosure_list`) with a `print()` method.

## See Also

`diagnose()` for utility.

## Examples

```
df <- data.frame(
  id  = 1:200,
  age = round(rnorm(200, 60, 10)),
  sex = sample(c("F", "M"), 200, replace = TRUE)
)
res <- synth(df, ~ id, seed = 1)
disclosure_risk(df, res, quasi = c("age", "sex"))
```

---

# dp_control *Differential-privacy controls (Track B)*

---

## Description

Opt into differentially private synthesis. Passing the result as `synth(..., privacy = dp_control(...))` selects the DP synthesiser and yields a formal $(\epsilon, \delta)$ guarantee. The privacy unit is **person-level** by default: the guarantee protects a whole individual (all of their rows), enforced by bounding each person's contribution before any budget is spent.

## Usage

```
dp_control(
  epsilon,
  delta = 0,
  unit = c("person", "row"),
  max_rows_per_person = NULL,
  mechanism = c("laplace", "gaussian"),
  dependence = c("tree", "independent"),
  structure_frac = NULL,
  degree = 1L,
  select = c("fixed", "adaptive", "aim"),
  treewidth = 1L,
  select_frac = 0.25,
  anneal = FALSE,
  estimator = c("local", "pgm"),
  scoring = c("auto", "independence", "model"),
  cross_table = FALSE,
  longitudinal = FALSE,
  baseline = NULL,
```

```
  transition_order = 1L,
  transition_cross = 0L,
  transition_parent = 0L,
  bins = 12L,
  bounds = NULL,
  domain = c("dp", "public", "data"),
  domain_frac = 0.1
)
```

## Arguments

| | |
|---|---|
| epsilon | Positive privacy-loss budget. Smaller = more private, less utility. |
| delta | Failure probability for approximate DP; `0` requests pure $\epsilon$-DP (only the Laplace mechanism supports `delta = 0`). |
| unit | Privacy unit. `"person"` (default) protects an individual across all their rows; `"row"` is weaker and cheaper (each row independent). |
| max_rows_per_person | Cap on how many rows one person may contribute, used to bound sensitivity at `unit = "person"`. `NULL` (default) uses `1` (each person contributes at most one row); rows beyond the cap are dropped by per-person subsampling before measuring. Set this from public domain knowledge when a person legitimately has several rows. For a longitudinal DP release (a `structure` with a nesting index), this must be `>= 2` — it caps the transition sensitivity — and the per-person prefix of that many rows is kept in temporal order. For a linked DP release (`synth_linked()`) it is the maximum children kept per parent — a single integer applied to every child table, or a **named** vector/list keyed by child-table name (e.g. `list(admissions = 8, labs = 20)`) — and the root cap is always 1. |
| mechanism | Noise mechanism: `"laplace"` (pure $\epsilon$-DP) or `"gaussian"` (approximate DP with zCDP composition; needs `delta > 0`). |
| dependence | Dependence structure of the generative model: `"tree"` (default; a Chow-Liu tree over pairwise marginals) or `"independent"` (one-way marginals only). |
| structure_frac | Budget-efficient structure learning for the flat `dependence = "tree"` release (`synth()` with three or more variables). `NULL` (default) measures all $\binom{d}{2}$ pairwise marginals at full fidelity and reuses them as both the Chow-Liu weights and the tree's conditionals. A number in `(0, 1)` instead spends only that fraction of the marginal budget on a cheap all-pairs scan used **solely** to select the tree, then concentrates the remaining `1 - structure_frac` on re-measuring just the $d-1$ chosen edges (plus the `d` one-way marginals) — so the budget lands on the parameters that survive into the model instead of on pairs that are discarded. Structure selection tolerates noise well, so a small slice (e.g. `0.2`–`0.3`) usually sharpens the retained edges, more so as the number of variables grows. Both passes compose into the same exact ($\epsilon$, $\delta$) budget. Inert for `dependence = "independent"`, for fewer than three variables (the tree is then trivial), and for longitudinal / linked DP releases (which keep the single-pass fitter). |
| degree | Fan-in of the Bayesian network fitted for a flat `synth()` `dependence = "tree"` release. `1` (default) keeps the Chow-Liu tree exactly as before (each variable |

conditions on at most one parent). `2` or more opts into **PrivBayes' Greedy-Bayes**: a degree-k Bayesian network in which each variable may condition on up to `degree` of the already-generated variables, its parent set chosen greedily with the **exponential mechanism** (the parents whose joint the variable depends on most, crediting the parents→child association only). Because a tree gives a variable a single parent, it cannot represent a variable that genuinely depends on two otherwise-unrelated predecessors (a "v-structure", e.g. `C` a noisy XOR of independent `A` and `B`); a degree-2 network takes both parents and recovers it. Construction measures the `d` one-way marginals plus one `(parents, node)` family joint per non-root node (`2d - 1` marginals total) and spends a `select_frac` slice on the `d - 1` greedy picks; every slice composes into the same exact ($\epsilon$, $\delta$) (zCDP for Gaussian, pure $\epsilon$ for Laplace). The network is forward-sampled ancestrally (no PGM inference), so like the tree it stays inference-free. `degree` is capped to `d - 1`, and a high fan-in makes each family a `(degree + 1)`-way histogram — the same `bins^(degree + 1)` cell-count warning as `treewidth` applies. Requires `dependence = "tree"`; it is an **alternative** to `structure_frac` and to `select = "adaptive"` (setting either alongside `degree > 1` is an error), and is refused on longitudinal / linked releases. Ignored when `degree = 1`.

`select` How the marginals that make up the model are chosen, for a flat `synth()` release. `"fixed"` (default) measures a predetermined set (all one-way marginals, plus — for `dependence = "tree"` — pairwise marginals), exactly as described above; every existing path is unchanged. `"adaptive"` opts into an **AIM-style** iterative selector: after measuring the one-way marginals it builds the dependency model one marginal at a time, at each step using the **exponential mechanism** to privately pick the marginal whose true value the model-so-far fits worst, then measuring that marginal under the main mechanism. Selection and measurement each spend budget and compose exactly into the same ($\epsilon$, $\delta$) (zCDP for Gaussian, pure $\epsilon$ for Laplace); the number of rounds is fixed in advance from the variable count and `treewidth`, so the accounting is data-independent. Unlike `structure_frac` (which picks a tree from one free noisy scan), adaptive selection is model-error-guided *and* — at `treewidth >= 2` — can measure marginals that form loops, capturing higher-order interactions a Chow-Liu tree structurally cannot. Ignores `dependence` and `structure_frac` (supply `treewidth` / `select_frac` instead — setting `structure_frac` alongside is an error). `"aim"` is **Full AIM**: it lifts the adaptive selector's running-intersection constraint, so the exponential mechanism may pick *loopy* marginals — a pair between two variables already in the model, the cycle a junction-tree selector structurally cannot close. Because a loopy set has no forward junction-tree sampler, `"aim"` reconciles the whole measured set (the `d` one-way marginals plus the selected pairs) into one graphical model over a **triangulated** junction tree by **Private-PGM** (belief propagation + mirror descent; see `estimator`) and samples from that. It runs a data-independent `min(d (d - 1) / 2, treewidth * (d - 1))` selection rounds, each rejecting any new pair whose triangulated clique would exceed `treewidth + 1`; at `treewidth = 1` no loop can be closed, so it reduces to an adaptively-selected tree. Like `"adaptive"` it uses `treewidth` / `select_frac` (not `structure_frac` / `degree`), and the reconciliation is budget-neutral. All selectors are currently **flat-table only**: `"adaptive"` / `"aim"` on a longitudinal or linked structure is refused.

`treewidth` Adaptive selection only. Maximum clique size minus one in the junction-tree model — the ceiling on interaction order the model can hold. `1` (default) builds a tree (each measured marginal is a pair; equivalent *model class* to `dependence = "tree"`, but selected adaptively). `2` lets the selector measure three-way marginals whose variables form triangles, so pairwise-invisible three-way structure survives, and `3` measures four-way cliques (interactions no three-way marginal can see, e.g. a 3-bit parity). Higher clique orders cost more per cell — a `(w+1)`-way histogram over `bins` cells per axis is `bins^(w+1)` cells, so the per-cell DP noise grows fast; a warning is raised when that count is large (`bins` is the numeric proxy; low-cardinality factor cliques are smaller). Raise it only when interactions of that order genuinely matter. For `select = "aim"` it bounds the model's *triangulated* clique size the same way (so `1` forbids loops and `2` permits triangles). Ignored unless `select = "adaptive"` or `select = "aim"`.

`select_frac` Adaptive selection only. Fraction of the marginal budget spent on the private **selection** (the exponential-mechanism rounds); the remaining `1 - select_frac` is spent **measuring** the chosen marginals (and the one-way marginals). Default `0.25`. Selection tolerates noise well, so a modest slice usually gives the best fidelity (the same intuition as `structure_frac`); both slices compose into the same exact ($\epsilon$, $\delta$). Ignored unless `select = "adaptive"` or `select = "aim"`.

`anneal` Adaptive (`"adaptive"` / `"aim"`) selection only. `FALSE` (default) uses a fixed-round schedule with a uniform per-round budget. `TRUE` opts into **AIM-style budget annealing** — a data-adaptive round schedule: the `d` one-way marginals take a fair fixed share, then the marginal measurements and their exponential-mechanism selections start at a small per-round quantum (large noise) and **double** whenever a round's measured signal fails to beat its noise floor (AIM's $\sigma$-halving rule). After a baseline number of selection rounds — the spanning cliques for `"adaptive"`, the treewidth-capped new pairs for `"aim"` — any **surplus** budget is spent on extra rounds that re-measure the worst-fit already-measured marginal (inverse-variance combined, so $\rho$ / $\epsilon$ adds exactly). The final round absorbs the exact remainder, so the total spend is still exactly ($\epsilon$, $\delta$) — now over a **variable** number of rounds chosen from the data. For `select = "adaptive"` the model stays a spanning junction tree (no PGM inference), so refinement can only sharpen the measured cliques; for `select = "aim"` the annealed set is triangulated and reconciled with Private-PGM exactly as in the fixed-round AIM, so refinement sharpens loopy marginals too. It is most useful when there are few variables and the fixed schedule would otherwise leave budget on the table. Ignored unless `select = "adaptive"` or `select = "aim"`; setting it otherwise is an error.

`estimator` How the measured marginals are turned into the generative model, for a flat `synth()` tree or adaptive release. `"local"` (default) is every existing path unchanged: each marginal is used **locally** — the tree takes the root's one-way and each edge's raw noisy 2-way as $P(\text{child} \mid \text{parent})$, the adaptive junction tree each clique's own array — so the other one-way marginals, and the disagreement between overlapping noisy marginals, are discarded. `"pgm"` opts into **Private-PGM inference** (McKenna et al.'s MST / graphical-model estimation): the *whole* set of measured marginals is reconciled into the single graphical-model distribution that best fits all of them at once (least squares), by belief propagation on a junction tree of the measured cliques plus entropic mirror descent,

and the model is sampled from that. Because a tree (or the adaptive junction tree) has bounded treewidth, the inference is exact and cheap. Reconciliation is **pure post-processing** of the already-privatised marginals, so it spends **no extra budget** — the ($\epsilon$, $\delta$) is identical to the same release with `"local"`; only the fitted model changes. It denoises (overlapping marginals are made mutually consistent) and lets the otherwise-discarded one-way marginals actually constrain the model, usually sharpening both the marginals and the conditionals at the same budget. Available for the flat `dependence = "tree"` release (degree-1) and for `select = "adaptive"`; it is an **alternative** to `structure_frac`, to `degree > 1`, and to `anneal = TRUE` (setting any of those alongside `estimator = "pgm"` is an error), and there is nothing to reconcile in `dependence = "independent"`. Refused on longitudinal / linked releases (flat-table only). Ignored (no-op) for `"local"`. `select = "aim"` always reconciles with Private-PGM by construction (a loopy marginal set has no local sampler), so `estimator` is moot there.

`scoring` How `select = "aim"` scores each candidate marginal in its exponential-mechanism rounds. `"auto"` (default) picks the faithful choice per selector: `select = "aim"` **uses** `"model"` (AIM's actual quality function), and every other selector uses `"independence"` (where model scoring is inert). `"model"` scores each candidate against the **current reconciled Private-PGM model's own marginal** over that pair, so a loopy pair the model already explains (through the marginals measured so far) no longer looks surprising and the budget is steered to the genuinely worst-fit interaction. `"independence"` is the cheaper opt-out: it scores a pair by how far its true joint sits from the **product of its one-way marginals** — the reference the other selectors use, and Full AIM's behaviour before this became the default. The model reference is read from the already-privatised marginals — reconciled each round and projected onto the candidate (a not-yet-measured pair projects across cliques of the junction tree) — so it is **pure post-processing**: the exponential mechanism's sensitivity and the exact ($\epsilon$, $\delta$) are **identical** to `"independence"`; only which marginals get selected changes. `"model"` costs extra computation (a reconciliation per selection round) and composes with `anneal`. Explicit `"model"` requires `select = "aim"` (an error otherwise); the adaptive selector's candidates always introduce a fresh variable, so its model projection reduces to the independence product anyway.

`cross_table` Linked DP only (`synth_linked()`). When `TRUE`, a child table's variables are conditioned on the **synthetic parent's** attributes: for each child table with a modellable immediate parent, parent-by-child joint marginals are measured (at the child's per-entity path-cap sensitivity, the same as a child one-way marginal) and folded into the child's Chow-Liu structure as fixed context nodes, so each child variable's single strongest predictor may be a parent variable or another child variable. The synthetic parent's already-drawn value then conditions the child draw, so cross-table statistical dependence — not just referential integrity — survives the noise. Costs extra budget (the added joints compose into the same ($\epsilon$, $\delta$)); `FALSE` (default) keeps child variables on their own within-table marginals. Ignored by flat / longitudinal `synth()` releases, which have no parent table.

`longitudinal` Linked DP only (`synth_linked()`). Model a child table's repeated rows as a within-unit time series under DP, instead of exchangeable records: for such a table the children-per-parent count model doubles as a trajectory-length model, an

initial-state model is measured over each unit's first (temporally earliest) child row, and a first-order Markov transition matrix $P(v_t \mid v_{t-1})$ is measured per variable over consecutive within-unit rows (ordered by the child's own key index). Sensitivities differ by histogram — initial marginals at the parent's path cap, transitions at `path_cap[parent] * (branching_cap - 1)` — and fold into the same exact ($\epsilon$, $\delta$) budget. Pass `TRUE` to model every eligible child table (a child with a branching cap $\geq 2$ and at least one variable), or a character vector of child-table names to model only those. The child's branching cap in `max_rows_per_person` bounds the transition sensitivity, so it must be $\geq 2$ for a named table, and the prefix of that many rows is kept in temporal order. Setting `cross_table = TRUE` together with a longitudinal model on the same table **combines** them: the table's initial-state model is cross-conditioned on the synthetic parent (adding the parent-by-child joints at the same first-row sensitivity as the initial marginals), and the transition chain then carries that parent dependence across the trajectory — the parent shapes where a trajectory starts, while the transitions stay parent-free, so the only extra cost is the initial-state joints. `FALSE` (default) treats child rows as exchangeable. Ignored by flat / longitudinal `synth()` releases (which pick the DP Markov engine straight from the `structure` formula).

`baseline` Longitudinal releases. Names of **subject-invariant** columns — baseline covariates that do not change across a unit's rows (e.g. birth sex, a baseline measurement). These are held **exactly constant** within each synthetic unit: they are modelled once in the initial-state model (so their joint distribution and their correlation with the first visit are preserved) and then broadcast to every row, rather than being stepped through a transition matrix that would let them drift. Declaring a column baseline is public schema knowledge, so it costs no budget; it also **removes** that column's transition histogram from the release, sharpening every remaining measurement at the same ($\epsilon$, $\delta$). Applies to a longitudinal `synth()` release (a `structure` with a nesting index) and, per table, to any **longitudinally-modelled linked child** (the names are matched against each such child's own columns). `NULL` (default) treats every column as time-varying. Names that match no modelled column are ignored. Ignored by a flat `synth()` release (one row per unit) and by non-longitudinal linked child tables.

`transition_order`

Longitudinal releases. Markov order of the within-unit transition model: how many of a variable's **own** immediately preceding values condition its next value. `1` (default) is the first-order model $P(v_t \mid v_{t-1})$; `2` gives $P(v_t \mid v_{t-1}, v_{t-2})$, and so on. Higher orders capture momentum but need a larger row cap: the order must be at most `max_rows_per_person - 1` — for a linked release, one less than the **branching cap** of each longitudinally-modelled child (validated per table). A unit then contributes at most `cap - order` transition tuples per variable, so the transition sensitivity — and hence the noise — actually *drops* with order, at the price of a finer conditioning grid (sparser cells) and of not modelling the earliest rows' own dynamics separately: rows before position `order + 1` are generated by marginalising the same measured tensor, which is post-processing and costs no budget. Applies to a longitudinal `synth()` release and, uniformly, to every longitudinally-modelled linked child; ignored by flat releases and non-longitudinal linked children.

`transition_cross`

Longitudinal releases. Number of **other** variables (each at lag 1) that additionally condition each variable's transition — moving from $P(v_t \mid v_{t-1}, \dots)$ to also conditioning on $u_{t-1}$ for the `transition_cross` most strongly associated companions `u`. Cross-parents are selected automatically and budget-neutrally from the pairwise marginals the Chow-Liu tree already measures (contemporaneous mutual information, a leak-free proxy for lag-1 cross-predictiveness) — for a linked child that means its own within-child pairwise marginals — so `transition_cross > 0` requires `dependence = "tree"`. Adding conditioning columns does **not** change the ($\epsilon$, $\delta$) budget — a transition tuple still lands in exactly one cell — it trades budget-free structure for cell sparsity, so keep it small. `0` (default) conditions each variable on its own past only. Applies to a longitudinal `synth()` release and to every longitudinally-modelled linked child; ignored by flat releases and non-longitudinal linked children.

`transition_parent`

Linked DP only (`synth_linked()`), longitudinal children. Number of **immediate-parent** attributes that additionally condition each time-varying child variable's transition. Today `cross_table = TRUE` cross-conditions only a longitudinal child's *initial state* on the synthetic parent; the parent's influence then rides the own-lag chain and decays. `transition_parent = p` instead re-injects the (subject-invariant) parent attributes into the transition tensor at **every** step — moving from $P(v_t \mid v_{t-1}, \dots)$ to also conditioning on the parent's $w$ for the `p` parent attributes `w` most strongly associated with `v` — so parent $\rightarrow$ child dependence stays anchored across the whole trajectory rather than washing out. The parents are selected automatically and **budget-neutrally** from the parent-by-child joints that the cross-conditioned initial state already measures, so it costs nothing in the ($\epsilon$, $\delta$) budget (a transition tuple still lands in exactly one cell — same sensitivity, same histogram count), trading budget-free structure for cell sparsity. Because it reuses those joints, `transition_parent > 0` **requires** `cross_table = TRUE` for the child (an error is raised otherwise). `0` (default) leaves the transitions parent-free (the initial-state cross-conditioning is unaffected). Composes with `baseline`, `transition_order` and `transition_cross`. Ignored by flat `synth()` releases and non-longitudinal linked children, which have no parent trajectory to condition.

`bins`

Number of equal-width bins used to discretise each numeric variable (default 12). Finer grids sharpen one-way marginals but make the noisy two-way marginals used by `dependence = "tree"` weaker per cell, so a moderate value usually gives the best correlation fidelity. Means and sums are largely unaffected (bin contents are decoded uniformly within the bin).

`bounds`

Optional named list giving `c(lower, upper)` for numeric variables, used as public, data-independent bin edges. How variables *not* named here are handled depends on `domain`.

`domain`

How the discrete domain is chosen for variables without public metadata: `"dp"` (default) estimates numeric bin edges — and discovers bare `character` category sets by DP set-union — under differential privacy and accounts for the cost; `"public"` requires `bounds` for every numeric variable and public `factor` levels for every categorical (error otherwise) and spends no budget on the domain; `"data"` reads edges / levels from the data with a warning and *excludes* that step

from the accounting (non-rigorous; benchmarking only).

`domain_frac` Fraction of the privacy budget spent learning the domain under `domain = "dp"` (default `0.1`) — numeric bin edges (exponential-mechanism quantiles) and `character` category sets (DP set-union). The slice is shared across those operations and composed with the marginal measurements, so the total spend is exactly ($\epsilon$, $\delta$). Ignored unless some numeric variable needs estimating or some `character` column needs discovering.

## Details

The DP engine (Track B) is a *marginal-based* synthesiser in the PrivBayes / MST lineage. Continuous variables are discretised into a public grid; low-order marginals are measured under the chosen noise mechanism with correct budget composition; and synthetic records are drawn from the resulting model. With `dependence = "tree"` a Chow-Liu tree of pairwise dependencies is learned from the same noisy marginals (no extra budget) so second-order structure is retained; `dependence = "independent"` keeps only one-way marginals. All noise calibration and composition is reported back on the result (see the accounting printed by `synth()`).

Where the bin edges come from matters, because reading them from the data (its min / max) is itself a data-dependent step that can leak an individual's presence. `domain` controls this:

`"dp"` **(default)** Rigorous and automatic. Numeric variables named in `bounds` use those public edges at no cost; any other numeric variable has its working range **estimated under differential privacy** (a clamp-free exponential-mechanism quantile at each end). A bare `character` column has its **category set discovered under differential privacy** by DP set-union (a stability histogram: each present category's noisy count must clear a threshold that hides any category a single person could have created; rare categories fold into an `"(other)"` catch-all). Both estimations spend an accounted slice `domain_frac` of the budget, so the reported ($\epsilon$, $\delta$) is exact end to end. DP set-union needs `delta > 0` (a threshold cannot hide a lone category's presence at `delta = 0`), so a pure-$\epsilon$ release still refuses `character`.

`"public"` Fully data-independent and free: every numeric variable **must** be given a public range in `bounds` (an error is raised otherwise) and no budget is spent on the domain. The recommended mode when public ranges (codebook / physiological limits) are available.

`"data"` The non-rigorous legacy behaviour: bin edges are taken from the data range with a warning, and that step is **excluded** from the accounting. Kept only for benchmarking; do not use for a governed release.

Categorical variables carry their domain in their type: `factor` and `logical` columns use their declared levels (public metadata, no leakage). In the `"dp"` and `"public"` modes a bare `character` column is refused — convert it to a `factor` with its full `levels` so the category set is public.

## Value

An object of class `dp_control` (a validated list).

## Examples

```
dp <- dp_control(epsilon = 1, delta = 1e-6, mechanism = "gaussian")
dp
```

## list_methods *List the available synthesis methods*

### Description

Return the names of every method currently registered (built-in plus any added with `register_method()`).

### Usage

```
list_methods()
```

### Value

A character vector of method names.

### Examples

```
list_methods()
```

## plot.flexsynth_diagnostics *Plot utility diagnostics*

### Description

Overlays the real and synthetic marginal distribution of each variable (density for numeric, side-by-side bars for categorical) using base graphics.

### Usage

```
## S3 method for class 'flexsynth_diagnostics'
plot(x, vars = NULL, max_panels = 12L, ...)
```

### Arguments

| | |
|---|---|
| `x` | A `flexsynth_diagnostics` object from `diagnose()`. |
| `vars` | Optional subset of variables to plot. |
| `max_panels` | Maximum number of variables to draw (default 12). |
| `...` | Passed to the underlying plotting calls. |

### Value

`x`, invisibly.

pool_synth *Pooled inference from synthetic data*

## Description

Fit an analysis on each of the `m` synthetic datasets in a `synth()` result and combine the results into one estimate whose standard error reflects the extra variability that synthesis introduces. An analysis of a single synthetic dataset does not estimate between-synthesis variation, so its usual standard errors can be miscalibrated. These functions implement published fully-synthetic combining rules; their large-sample calibration still depends on the estimand, synthesis model, sample size, and analysis assumptions.

## Usage

```
pool_synth(
  object,
  analysis,
  rule = c("synthpop", "reiter"),
  population_inference = TRUE,
  conf.level = 0.95,
  ...
)
```

## Arguments

| | |
|---|---|
| `object` | A `synth_result` from `synth()` (ideally with `m >= 5`). |
| `analysis` | A function of one synthetic `data.frame` returning a fitted model (with `coef()`/`vcov()`) or a `list(estimate =, variance =)`. |
| `rule` | Combining rule: `"synthpop"` (default, Raab/Nowok/Dibben 2016) or `"reiter"` (Reiter 2003 fully-synthetic; needs `m >= 2`). |
| `population_inference` | If `TRUE` (default) the standard error targets the population the real data was drawn from, inflating for synthesis. If `FALSE` it reports only the (size-rescaled) within-synthesis variance — the standard error an analyst of the original data would have seen. |
| `conf.level` | Confidence level for the intervals (default `0.95`). |
| `...` | Reserved for future use. |

## Details

`analysis` is run once per synthetic dataset and must return either a fitted model supporting `coef()` and `vcov()` (such as `lm()` or `glm()`) or a list with numeric `estimate` and `variance` (squared standard error) vectors. The per-dataset estimates are then combined with the chosen `rule`: `"synthpop"` (Raab, Nowok & Dibben 2016, matching synthpop's `summary.fit.synds`) or `"reiter"` (Reiter 2003 fully-synthetic, `Tf = (1 + 1/m) b_m - vbar`, which needs `m >= 2` and can be negative). Intervals are large-sample normal, as in synthpop.

Only Track A (synth() without privacy) results are supported: differentially private (Track B) inference must additionally account for the DP noise and is out of scope here. Pooling of a synth_linked() result is not yet supported; analyse a single table's synthetic frame instead.

## Value

A flexsynth_pool object; $estimates is a data frame with term, estimate, std.error, statistic, p.value, conf.low, conf.high.

## See Also

synth_glm() for the common linear / generalised-linear case.

## Examples

```
d <- data.frame(id = 1:400, x = rnorm(400))
d$y <- 1 + 2 * d$x + rnorm(400)
res <- synth(d, ~ id, m = 5, seed = 1)
pool_synth(res, function(dat) lm(y ~ x, dat))
```

---

# register_method *Register a synthesis method*

---

## Description

Add a custom per-variable synthesiser to the method registry so it can be selected by name via the method argument of synth() / synth_linked() or the per-variable method in synth_control(). Registering a name that already exists overwrites it, so you can override a built-in.

## Usage

```
register_method(
  name,
  fit,
  draw,
  numeric = TRUE,
  categorical = TRUE,
  needs_predictors = TRUE
)
```

## Arguments

| | |
|---|---|
| name | Method name (a single string) used to select it. |
| fit | A function function(y, x, control) returning a fitted model. |
| draw | A function function(model, x, n, control) returning n values. |
| numeric, categorical | Logical flags declaring which target types the method supports (used for validation). Both default to TRUE. |

needs_predictors
: Logical; if TRUE (default) the engine falls back to marginal sampling when the variable has no predictors.

## Details

A method is a pair of functions:

`fit(y, x, control)` fits a model of the target `y` on the predictor `data.frame` `x` (which may have zero columns) and returns any object;

`draw(model, x, n, control)` returns a length-`n` vector of synthetic values for new predictor rows `x`.

`control` is the `synth_control()` object, so a method can honour `proper`, read its own hyperparameters, and so on.

## Value

Invisibly, the method name.

## Examples

```
# A trivial "mean" method for numeric variables.
register_method(
  "constant_mean",
  fit  = function(y, x, control) mean(y),
  draw = function(model, x, n, control) rep(model, n),
  categorical = FALSE
)
"constant_mean" %in% list_methods()
```

---

# rule *Declare a synthesis constraint*

---

## Description

Capture a logical rule the synthetic data must satisfy. Rules are enforced by `synth()` via rejection sampling at the unit grain (see *Details*).

## Usage

```
rule(expr, scope = c("row", "unit"), label = NULL)
```

### Arguments

| | |
|---|---|
| expr | A logical expression over the column names, e.g. dbp <= sbp. With scope = "row" it is evaluated across a unit's rows and must be TRUE for every row. With scope = "unit" it is evaluated once per unit (the columns are that unit's rows in positional order) and must return TRUE — use this for temporal logic such as all(diff(los) >= 0). Missing (NA) results are treated as satisfied. |
| scope | "row" (default) or "unit". |
| label | Optional human-readable label; defaults to the deparsed expression. |

### Details

The expression is written in terms of the data's column names and is **not** evaluated when rule() is called — it is captured and checked later against the synthetic data.

### Value

A flexsynth_rule object.

### Examples

```
# A row rule and a within-unit temporal rule.
rule(dbp <= sbp)
rule(all(diff(visit_time) > 0), scope = "unit")
```

---

## synth *Synthesise a single (optionally nested / longitudinal) table*

---

### Description

Generate synthetic data for one table, working natively in long format. The structure formula declares the nesting hierarchy (e.g. ~ id / visit / test_number) so repeated-measures and nested designs are kept without pivoting to wide format.

### Usage

```
synth(
  data,
  structure,
  method = "cart",
  constraints = NULL,
  tuning = synth_control(),
  privacy = NULL,
  m = 1,
  seed = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| data | A data.frame in long format. |
| structure | A one-sided formula giving the nesting hierarchy, e.g. ~ id / visit. The first term is the unit identifier. |
| method | Synthesis method, "cart" (default) or "sample", applied per variable unless overridden in tuning. |
| constraints | Optional rule() (or list of rules) the synthetic data must satisfy. Enforced by rejection sampling at the unit grain (see rule()); constraint_max_tries in synth_control() bounds the effort. |
| tuning | A synth_control() object. |
| privacy | NULL for Track A, or a dp_control() object for Track B. |
| m | Number of synthetic datasets to produce. |
| seed | Optional integer seed for reproducibility. |
| ... | Reserved for future use. |

## Details

The default (Track A) engine is sequential conditional synthesis in the synthpop lineage: the unit identifier is regenerated by resampling whole units (preserving realistic per-unit block sizes), and each remaining variable is synthesised in turn from a model fitted on the real data, conditioning on the structural columns and everything synthesised before it. With method = "cart" each draw comes from the matching leaf of a regression / classification tree, so marginal and conditional distributions are preserved without distributional assumptions. Track A carries **no** formal privacy guarantee.

Columns that are constant within every real unit (baseline covariates such as age or sex) are detected automatically and synthesised **once per unit**, then broadcast across that unit's rows, so a synthetic subject stays internally consistent. Time-varying columns are synthesised with an initial-state model (first row of each unit) plus a Markov transition model that conditions on the previous row within the unit (lag-1 predictors), so autocorrelation across visits is preserved. The number of rows per unit is drawn from the learned count distribution and the structural-index sequence is regenerated for each synthetic unit.

Any column containing NAs is given a missingness model: a companion indicator is synthesised in sequence just before the column, the column's own value model is fitted on the observed rows only, and the synthesised indicator decides which rows are set back to NA. This preserves the missingness rate and its association with the other variables, without letting missingness cascade through predictors. The indicators are not returned.

Supplying constraints (see rule()) keeps only synthetic units whose rows satisfy every rule, regenerating until enough valid units are collected — so logical and within-unit temporal constraints are honoured without breaking the nested structure.

Passing privacy = dp_control(...) selects differentially private synthesis (Track B) with a formal ($\epsilon$, $\delta$) guarantee. A flat structure = ~ id gives a marginal release; a nesting index (~ id / visit) additionally engages a DP Markov model that preserves within-unit temporal structure (length, initial state, and per-variable transitions), for which max_rows_per_person must be set to the public maximum rows per person. See vignette("differential-privacy").

## Value

A synth_result object. Use as.data.frame() (for m == 1) or $syn to get the synthetic data.

## Examples

```
df <- data.frame(
  id    = rep(1:20, each = 2),
  visit = rep(1:2, times = 20),
  age   = rep(round(rnorm(20, 60, 8)), each = 2),
  sbp   = round(rnorm(40, 130, 15))
)
res <- synth(df, structure = ~ id / visit, seed = 1)
head(as.data.frame(res))
```

---

# synth_control *Tuning controls for synthesis*

---

## Description

Collects the fine-grained knobs for the sequential synthesis engine. Strong defaults mean beginners can ignore this; power users can tune the utility / privacy / speed trade-off. Returns a validated synth_control object consumed by synth() and synth_linked().

## Usage

```
synth_control(
  visit_sequence = NULL,
  predictor_matrix = NULL,
  method = NULL,
  smoothing = NULL,
  proper = FALSE,
  count_model = c("marginal", "conditional"),
  k = NULL,
  cart = list(),
  forest = list(),
  constraint_max_tries = 50L,
  parallel = FALSE
)
```

## Arguments

| | |
|---|---|
| visit_sequence | Optional character vector giving the order in which variables are synthesised. NULL uses the natural (left-to-right) order. |
| predictor_matrix | Optional 0/1 matrix marking, for each variable, which variables may predict it (including cross-table predictors). NULL lets the engine derive a sensible default. |

`method` Optional per-variable method override; a single string applies globally. `NULL` uses the call-level `method`.

`smoothing` Numeric-variable kernel smoothing. `NULL` (default) is **auto**: smooth numeric variables that look continuous (more than 20 distinct values), so their draws are not confined to the observed values — improving marginal realism and reducing exact-value replication. `TRUE` / `"density"` smooths every numeric variable; `FALSE` / `"none"` disables smoothing; a character vector smooths exactly the named variables. Auto resolution applies to single-table `synth()`; `synth_linked()` children are unsmoothed unless named explicitly.

`proper` Logical; use proper (`TRUE`) vs improper (`FALSE`) synthesis.

`count_model` How the number of rows per unit is drawn for nested / longitudinal data. `"marginal"` (default) resamples the observed unit sizes independently of any covariate. `"conditional"` synthesises the subject-level covariates first and draws each unit's size from a CART model of size on those covariates, so a dependence such as "sicker patients have more visits" is preserved. Only takes effect when the structure is nested and there are subject-level covariates; ignored for flat single-row-per-unit data.

`k` Optional size of each synthetic dataset. `NULL` matches the input.

`cart`, `forest` Named lists of hyperparameters passed to the CART / random forest backends. For `forest`, `ntree` (number of trees, default 10) and `mtry` (predictors tried per tree, default all) are recognised.

`constraint_max_tries`
Integer; how many times `synth()` may regenerate while rejection-sampling to satisfy `constraints`. Defaults to 50.

`parallel` Controls parallel generation of the `m` synthetic replicates. `FALSE` (default) synthesises serially. `TRUE` uses all available workers (`getOption("mc.cores")`, else `parallel::detectCores()`); a positive integer sets an explicit worker count. Parallelism spreads the independent replicates over a cluster, so it only helps when `m` > 1. A parallel run is reproducible for a fixed (`seed`, worker count) via independent L'Ecuyer RNG streams, but its output differs from a serial run with the same seed.

## Value

An object of class `synth_control` (a validated list).

## Examples

```
ctrl <- synth_control(proper = TRUE, cart = list(minbucket = 5))
ctrl
```

synth_glm *Pooled generalised-linear analysis of synthetic data*

## Description

Convenience wrapper over pool_synth() for the common case: fit the same glm() on each synthetic dataset and combine. With the default family = gaussian() this is an ordinary linear model.

## Usage

```
synth_glm(
  object,
  formula,
  family = stats::gaussian(),
  rule = c("synthpop", "reiter"),
  population_inference = TRUE,
  conf.level = 0.95,
  ...
)
```

## Arguments

| | |
|---|---|
| object | A synth_result from synth(). |
| formula | A model formula (e.g. y ~ x1 + x2). |
| family | A family() for glm(); default gaussian() (linear model). |
| rule, population_inference, conf.level | Passed to pool_synth(). |
| ... | Further arguments to glm(). |

## Value

A flexsynth_pool object (see pool_synth()).

## Examples

```
d <- data.frame(id = 1:400, x = rnorm(400))
d$y <- rbinom(400, 1, plogis(-0.5 + d$x))
res <- synth(d, ~ id, m = 5, seed = 1)
synth_glm(res, y ~ x, family = binomial())
```

synth_linked *Jointly synthesise multiple linked nested / longitudinal tables*

## Description

Synthesise several related tables together so that referential integrity and cross-table statistical relationships are preserved. The table hierarchy is read from the keys: a table is a *child* of the table whose full key equals its own key with the last column dropped (so `c("id", "admission_id")` is a child of `c("id")`). Root tables are synthesised first with the single-table engine (`synth()`); each child table is then generated from its synthetic parent.

## Usage

```
synth_linked(
  tables,
  structures,
  keys,
  method = "cart",
  constraints = NULL,
  tuning = synth_control(),
  privacy = NULL,
  m = 1,
  seed = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| tables | Named list of input data.frames (one per table). |
| structures | Named list of one-sided formulas, one per table, giving each table's nesting hierarchy (e.g. ~ id / admission_id). |
| keys | Named list of character vectors giving each table's key columns. The last column is the table's own index; the leading columns are the foreign key into its parent. |
| method | Synthesis method applied per variable unless overridden. |
| constraints | Optional cross-table constraints / rules. Constraint enforcement currently applies to single-table `synth()` only; a message is emitted if supplied here (Track A) and it is an error under privacy (DP). |
| tuning | A `synth_control()` object. |
| privacy | NULL for Track A, or a `dp_control()` object for Track B. |
| m | Number of synthetic dataset collections to produce. |
| seed | Optional integer seed for reproducibility. |
| ... | Reserved for future use. |

## Details

For every synthetic parent record the number of child rows is drawn from a learned count distribution (including parents with **no** children), the child's foreign key is copied from the parent so it always resolves (referential integrity), the child's own structural index is regenerated, and the child's variables are synthesised conditionally on the parent's synthesised attributes (**cross-table predictors**), the own index and earlier child variables. Conditioning is on the immediate parent; a grandparent reaches a child only through the parent's synthesised values. Use `check_linkage()` to verify the result. Track A carries **no** formal privacy guarantee.

Passing `privacy = dp_control(...)` selects differentially private synthesis (Track B) with a formal ($\epsilon$, $\delta$) guarantee at the **root-entity** grain: adding or removing one root individual (its root row and all descendant rows) changes the release within the budget. Contribution is bounded hierarchically — `max_rows_per_person` gives the maximum children kept per parent for each child table (a single integer for all child tables, or a named list keyed by table name); the root cap is 1. Each table's variable marginals and a children-per-parent count histogram are measured under one exactly-composed budget; synthetic children copy the synthetic parent's surrogate key so referential integrity still holds by construction. By default a child's variables are modelled by their own within-table marginals (referential integrity is preserved, but not cross-table statistical dependence) and its repeated rows are treated as exchangeable. Two opt-ins in `dp_control()` add structure at extra, exactly-composed budget: `cross_table = TRUE` conditions each child variable on its synthetic parent's attributes (so parent $\rightarrow$ child dependence, not just referential integrity, survives the noise), and `longitudinal = TRUE` (or a character vector of child-table names) models a child's repeated rows as a within-unit DP Markov trajectory — an initial-state model plus per-variable $P(v_t \mid v_{t-1})$ transitions — so within-table longitudinal structure such as visit-to-visit autocorrelation is retained; the two combine. A longitudinally-modelled child needs a branching cap $\geq 2$ in `max_rows_per_person`, and its recoverable temporal signal falls as the noise grows (deeper nesting, larger caps, finer `bins`, or smaller $\epsilon$). `constraints` are refused under DP. See `vignette("differential-privacy")`.

## Value

A `synth_linked_result`. Use `as.list()` (for `m == 1`) or `$syn` to get the named list of synthetic tables.

## Examples

```
patients <- data.frame(id = 1:20,
                       sex = sample(c("F", "M"), 20, TRUE),
                       stringsAsFactors = FALSE)
adm <- do.call(rbind, lapply(patients$id, function(pid) {
  n <- 1 + rpois(1, 0.6)
  data.frame(id = pid, admission_id = seq_len(n),
             los = 1L + rpois(n, 3))
}))
res <- synth_linked(
  tables     = list(patients = patients, admissions = adm),
  structures = list(patients = ~ id, admissions = ~ id / admission_id),
  keys       = list(patients = "id", admissions = c("id", "admission_id")),
  seed = 1
)
```

```
check_linkage(res)
```

# Index