Package 'lingamr'

July 17, 2026

Type Package

Title 'LiNGAM' Algorithms for Causal Discovery

Version 0.1.2

Description R implementation of 'LiNGAM' (Linear Non-Gaussian Acyclic Model) algorithms for causal discovery, following Shimizu et al. (2011) <https://www.jmlr.org/papers/v12/shimizu11a.html>. Based on the 'Python' implementation by Ikeuchi et al. (2023) <https://github.com/cdt15/lingam>. The 'VAR-LiNGAM' residual diagnostics are inspired by the 'VARLiNGAM' R code of Moneta et al. <https://sites.google.com/site/dorisentner/publications/VARLiNGAM>.

License MIT + file LICENSE

URL https://github.com/morimotoosamu/lingamr, https://morimotoosamu.github.io/lingamr/

BugReports https://github.com/morimotoosamu/lingamr/issues

Depends R (>= 4.1.0)

Imports generics, grDevices, parallel, stats, utils

Suggests DiagrammeR, ggplot2, glmnet, igraph, knitr, lavaan, mice, nortest, pcalg, rmarkdown, spelling, testthat (>= 3.0.0), tseries

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

Language en-US

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Osamu Morimoto [aut, cre, cph], T. Ikeuchi [cph], G. Haraoka [cph], M. Ide [cph], W. Kurebayashi [cph], S. Shimizu [cph]

Maintainer Osamu Morimoto <galactic.supermarket@gmail.com>

Repository CRAN

Date/Publication 2026-07-17 13:20:07 UTC

Contents

`as_bootstrap_result` *Collapse an ImputationBootstrapResult into a BootstrapResult*

Description

`bootstrap_with_imputation()`'s result carries an extra `n_repeats` dimension (one causal-structure estimate per imputed dataset per bootstrap iteration), which the existing bootstrap analysis functions (`get_probabilities()`, `get_causal_direction_counts()`, `get_directed_acyclic_graph_counts()`, `get_causal_order_stability()`, `tidy()`) do not expect. This collapses that dimension by aggregating the `n_repeats` adjacency matrices of each iteration into one, producing a regular `lingam_direct_bootstrap()`-style `BootstrapResult`.

Usage

```
as_bootstrap_result(x, aggregate = c("median", "mean"))
```

Arguments

`x`	An `ImputationBootstrapResult`, as returned by `bootstrap_with_imputation()`.
`aggregate`	How to aggregate across the `n_repeats` dimension: `"median"` (default) or `"mean"`.

Value

A `BootstrapResult` (see `lingam_direct_bootstrap()`) with `total_effects = NULL` (total effects are not computed by `bootstrap_with_imputation()`); calling `get_total_causal_effects()` on it raises the usual "not computed" error.

Examples

```
set.seed(1)
sample6 <- generate_lingam_sample_6(n = 300, seed = 1)
X <- sample6$data
X$x5[sample.int(nrow(X), size = 30)] <- NA

if (requireNamespace("mice", quietly = TRUE)) {
  res <- bootstrap_with_imputation(X,
    n_sampling = 5L, n_repeats = 3L, seed = 42, verbose = FALSE
  )
  bs <- as_bootstrap_result(res, aggregate = "median")
  get_probabilities(bs)

  # get_total_causal_effects() is not available: total effects were never computed
  tryCatch(get_total_causal_effects(bs), error = function(e) conditionMessage(e))
}
```

`autoplot.LiMResult` *Plot the causal graph of a LiMResult with ggplot2*

Description

Draws the estimated causal structure of a LiM model as a ggplot2-based directed graph, exactly like `autoplot.LingamResult()`.

Usage

```
## S3 method for class 'LiMResult'
autoplot(
  object,
  threshold = 0,
  node_size = 16,
  node_color = "lightblue",
  label_edges = TRUE,
  label_pos = 0.35,
  ...
)
```

Arguments

`object`	Return value of `lingam_lim()` (a `LiMResult` object)
`threshold`	Coefficients with an absolute value at or below this are not treated as edges (default: 0)
`node_size`	Node size (default: 16)
`node_color`	Node fill color (default: "lightblue")
`label_edges`	Whether to display coefficient labels on edges (default: TRUE)
`label_pos`	Position of each coefficient label along its edge, as a fraction from the source (0) to the target (1). The default 0.35 places labels off-center (toward the source) so labels on crossing edges do not overlap near the midpoint.
`...`	Unused

Value

A ggplot object

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE) &&
    requireNamespace("igraph", quietly = TRUE)) {
  library(ggplot2)
  set.seed(1)
  dat <- generate_lim_sample(n = 300)
  model <- lingam_lim(dat$data, is_continuous = dat$is_continuous)
```

```
  autoplot(model)
}
```

`autoplot.LingamResult` *Plot the causal graph of a LingamResult with ggplot2*

Description

Draws the estimated causal structure as a ggplot2-based directed graph. Node positions are computed with igraph's hierarchical layout (sugiyama), so the causal flow is generally arranged from top to bottom. Because the output is a static image, it is stable in RMarkdown / Quarto. If you need an interactive HTML figure, use `plot_adjacency()` (DiagrammeR-based).

Usage

```
## S3 method for class 'LingamResult'
autoplot(
  object,
  threshold = 0,
  node_size = 16,
  node_color = "lightblue",
  label_edges = TRUE,
  label_pos = 0.35,
  ...
)
```

Arguments

`object`	Return value of `lingam_direct()` (a `LingamResult` object)
`threshold`	Coefficients with an absolute value at or below this are not treated as edges (default: 0)
`node_size`	Node size (default: 16)
`node_color`	Node fill color (default: "lightblue")
`label_edges`	Whether to display coefficient labels on edges (default: TRUE)
`label_pos`	Position of each coefficient label along its edge, as a fraction from the source (0) to the target (1). The default 0.35 places labels off-center (toward the source) so labels on crossing edges do not overlap near the midpoint.
`...`	Unused

Details

`autoplot()` is a ggplot2 generic, so you must load ggplot2 with `library(ggplot2)` before using it. Plotting requires ggplot2 and igraph.

Value

A ggplot object

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE) &&
    requireNamespace("igraph", quietly = TRUE)) {
  library(ggplot2)
  dat <- generate_lingam_sample_6()
  model <- lingam_direct(dat$data, reg_method = "ols")
  autoplot(model)
}
```

---

autoplot.MultiGroupLingamResult

Plot one group of a MultiGroupLingamResult with ggplot2

Description

Extracts a single group's result with `get_group_result()` and draws it like `autoplot.LingamResult()`, with the group shown in the subtitle. The causal order is shared across groups, but the coefficients (and therefore the plotted edges) are group specific.

Usage

```
## S3 method for class 'MultiGroupLingamResult'
autoplot(
  object,
  group = 1,
  threshold = 0,
  node_size = 16,
  node_color = "lightblue",
  label_edges = TRUE,
  label_pos = 0.35,
  ...
)
```

Arguments

`object`	Return value of `lingam_multi_group()` (a `MultiGroupLingamResult` object)
`group`	Which group to plot: a group name (character) or a 1-based index (default: 1)
`threshold`	Coefficients with an absolute value at or below this are not treated as edges (default: 0)
`node_size`	Node size (default: 16)
`node_color`	Node fill color (default: "lightblue")

label_edges	Whether to display coefficient labels on edges (default: TRUE)
label_pos	Position of each coefficient label along its edge, as a fraction from the source (0) to the target (1). The default 0.35 places labels off-center (toward the source) so labels on crossing edges do not overlap near the midpoint.
...	Unused

Value

A ggplot object

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE) &&
    requireNamespace("igraph", quietly = TRUE)) {
  library(ggplot2)
  mg <- generate_multi_group_sample()
  model <- lingam_multi_group(mg$data_list, reg_method = "ols")
  autoplot(model, group = 2)
}
```

---

autoplot.ParceLingamResult

Plot the causal graph of a ParceLingamResult with ggplot2

Description

Draws the estimated causal structure as a ggplot2-based directed graph, like `autoplot.LingamResult()`. Variable pairs whose adjacency-matrix entries are `NA` (unresolved order / suspected latent confounding) are drawn as dashed, arrowless segments.

Usage

```
## S3 method for class 'ParceLingamResult'
autoplot(
  object,
  threshold = 0,
  node_size = 16,
  node_color = "lightblue",
  label_edges = TRUE,
  label_pos = 0.35,
  ...
)
```

Arguments

object	Return value of lingam_parce() (a ParceLingamResult object)
threshold	Coefficients with an absolute value at or below this are not treated as edges (default: 0)
node_size	Node size (default: 16)
node_color	Node fill color (default: "lightblue")
label_edges	Whether to display coefficient labels on edges (default: TRUE)
label_pos	Position of each coefficient label along its edge, as a fraction from the source (0) to the target (1). The default 0.35 places labels off-center (toward the source) so labels on crossing edges do not overlap near the midpoint.
...	Unused

Value

A ggplot object

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE) &&
    requireNamespace("igraph", quietly = TRUE)) {
  library(ggplot2)
  dat <- generate_parce_sample(n = 500, seed = 42)
  model <- lingam_parce(dat$data)
  autoplot(model)
}
```

---

autoplot.RCDResult *Plot the causal graph of an RCDResult with ggplot2*

Description

Draws the estimated causal structure as a ggplot2-based directed graph, like autoplot.LingamResult(). Variable pairs suspected to share a latent confounder (NA entries in the adjacency matrix) are drawn as dashed, arrowless segments.

Usage

```
## S3 method for class 'RCDResult'
autoplot(
  object,
  threshold = 0,
  node_size = 16,
  node_color = "lightblue",
  label_edges = TRUE,
  label_pos = 0.35,
  ...
)
```

Arguments

object	Return value of lingam_rcd() (an RCDResult object)
threshold	Coefficients with an absolute value at or below this are not treated as edges (default: 0)
node_size	Node size (default: 16)
node_color	Node fill color (default: "lightblue")
label_edges	Whether to display coefficient labels on edges (default: TRUE)
label_pos	Position of each coefficient label along its edge, as a fraction from the source (0) to the target (1). The default 0.35 places labels off-center (toward the source) so labels on crossing edges do not overlap near the midpoint.
...	Unused

Value

A ggplot object

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE) &&
    requireNamespace("igraph", quietly = TRUE)) {
  library(ggplot2)
  confounded <- generate_rcd_sample(n = 300, seed = 1)
  model <- lingam_rcd(confounded$data)
  autoplot(model)
}
```

---

bootstrap_with_imputation

Bootstrap with Multiple Imputation for Direct LiNGAM

Description

Causal discovery on data containing missing values (NA). Each bootstrap resample (drawn with replacement, missing values retained) is multiply imputed into n_repeats complete datasets, and a common causal structure is jointly estimated across those datasets with lingam_multi_group() (Shimizu 2012), treating the imputed copies as "groups" sharing one causal order. R port of the Python lingam.tools.bootstrap_with_imputation().

Usage

```
bootstrap_with_imputation(
  X,
  n_sampling,
  n_repeats = 10L,
  imputer = NULL,
```

```
  cd_fit = NULL,
  prior_knowledge = NULL,
  apply_prior_knowledge_softly = FALSE,
  seed = NULL,
  verbose = TRUE
)
```

Arguments

X	A numeric matrix or data frame (n_samples x n_features). May contain NA. If X has no missing values, a warning suggests using lingam_direct_bootstrap() instead, and estimation proceeds anyway.
n_sampling	Number of bootstrap iterations (positive integer)
n_repeats	Number of imputed datasets generated per bootstrap sample (positive integer, default 10L). Ignored when a custom imputer is supplied; the number of datasets it returns is used instead.
imputer	NULL, or a function(X_boot) returning a list of complete (no-NA) numeric matrices, each with the same dimensions as X_boot. Defaults to multiple imputation via mice::mice(method = "norm").
cd_fit	NULL, or a function(X_list) returning list(causal_order = <integer vector, 1-based permutat Defaults to joint estimation via lingam_multi_group().
prior_knowledge	Prior knowledge matrix (NULL allowed). Only used when cd_fit = NULL; a warning is issued if supplied together with a custom cd_fit.
apply_prior_knowledge_softly	Apply prior knowledge softly (logical). Same restriction as prior_knowledge.
seed	Random seed (NULL allowed). Set once before the bootstrap loop; governs both the resampling and (via the global RNG) mice's imputation.
verbose	Whether to display progress (logical)

Details

Procedure: for each of n_sampling iterations, (1) resample X with replacement (missing values are retained), (2) impute the resample into n_repeats complete datasets, (3) jointly estimate one causal structure shared by all n_repeats datasets with lingam_multi_group(). This assumes the same causal structure underlies every imputed copy.

Default imputer. mice::mice(method = "norm") (Bayesian linear regression, multiple imputation by chained equations) is the closest standard R analogue of the upstream Python default (IterativeImputer(sample_posterior = TRUE)). The two do not produce numerically identical imputations.

Custom imputer / cd_fit. Supply your own imputation or causal-discovery routine by passing a function with the signature described above; the return value is validated and a descriptive error is raised on violation. This replaces the abstract base classes (BaseMultipleImputation, BaseMultiGroupCDModel) of the Python original.

Downstream analysis. The result's shape (an extra n_repeats dimension for adjacency_matrices and imputation_results) differs from lingam_direct_bootstrap()'s BootstrapResult, so it

cannot be passed directly to get_probabilities() etc. Use as_bootstrap_result() to collapse the n_repeats dimension (aggregating by median or mean) into a BootstrapResult.

On iteration failures: each iteration is wrapped in tryCatch(); a failing iteration (e.g. mice fails to converge on a particular resample) is skipped with a warning, and only if every iteration fails is an error raised, mirroring lingam_direct_bootstrap().

Sequential execution only. Unlike lingam_direct_bootstrap(), this function does not support parallel = TRUE; the upstream Python implementation is sequential as well. If needed in the future, it can be parallelized following the parallel::makePSOCKcluster() pattern used by lingam_direct_bootstrap().

Value

An ImputationBootstrapResult (list) containing:

- causal_orders: n_sampling x p integer matrix (1-based causal order per iteration).
- adjacency_matrices: array(n_sampling, n_repeats, p, p); [, , i, j] follows the lingamr convention (B[i, j] = coefficient of j -> i).
- resampled_indices: n_sampling x n integer matrix of resampled row indices.
- imputation_results: array(n_sampling, n_repeats, n, p); non-NA only at positions that were missing in that iteration's bootstrap resample.

Examples

```
set.seed(1)
sample6 <- generate_lingam_sample_6(n = 300, seed = 1)
X <- sample6$data
X$x5[sample.int(nrow(X), size = round(0.1 * nrow(X)))] <- NA # MCAR 10% on x5


if (requireNamespace("mice", quietly = TRUE)) {
  res <- bootstrap_with_imputation(X,
    n_sampling = 5L, n_repeats = 3L, seed = 42, verbose = FALSE
  )
  print(res)

  # Collapse the n_repeats dimension to reuse the existing bootstrap tooling
  bs <- as_bootstrap_result(res, aggregate = "median")
  get_probabilities(bs)
}
```

---

check_var_stationarity *Check the stationarity of a fitted VAR-LiNGAM model*

Description

Recovers the reduced-form VAR coefficients `M_k = (I - B0)^{-1} B_k` from the structural matrices and inspects the eigenvalues of the VAR companion matrix. The process is stationary when every eigenvalue lies strictly inside the unit circle (all moduli < 1); a modulus on or outside it signals a (near-)unit root, under which the VAR-LiNGAM estimates are unreliable.

Usage

```
check_var_stationarity(result, tol = 1)
```

Arguments

`result`	a `VARLiNGAMResult` from `lingam_var()`
`tol`	stationarity threshold for the eigenvalue moduli (default 1)

Value

a `var_stationarity` object (list) with `moduli` (sorted descending), `max_modulus`, `is_stationary` (logical), `lags`, and `tol`.

References

Stationarity diagnostics in the spirit of the VARLiNGAM R code of Moneta, A., Entner, D., Hoyer, P. O., & Coad, A. (2013), *Oxford Bulletin of Economics and Statistics*, 75(5), 705-730. https://sites.google.com/site/dorisentner/publications/VARLiNGAM

Examples

```
s <- generate_varlingam_sample(n = 1000, seed = 42)
m <- lingam_var(s$data, lags = 1, reg_method = "ols", prune = FALSE)
check_var_stationarity(m)
```

---

estimate_all_total_effects

Estimate the total causal effects between all variables at once

---

Description

Estimate the total causal effects between all variables at once

Usage

```
estimate_all_total_effects(
  X,
  lingam_result,
  method = "adaptive_lasso",
  lambda = "BIC",
  init_method = "ols"
)
```

Arguments

X	Original data (n_samples x n_features)
lingam_result	Return value of lingam_direct()
method	Regression method ("ols", "lasso", "adaptive_lasso", "ridge")
lambda	Lambda selection ("lambda.min", "lambda.1se", "AIC", "BIC")
init_method	Method for estimating the initial weights of adaptive LASSO regression ("ols" or "ridge")

Value

Matrix of total causal effects (n_features x n_features). **Convention:** `TE[i, j]` **is the total causal effect from variable j to variable i (j -> i).** Same index convention as the adjacency matrix `adjacency_matrix`. The sum of direct and indirect effects.

Examples

```
LiNGAM_sample_1000 <- generate_lingam_sample_6()

model <- LiNGAM_sample_1000$data |>
  lingam_direct(reg_method = "ols")

LiNGAM_sample_1000$data |>
  estimate_all_total_effects(model)
```

---

estimate_total_effect *Estimate the total causal effect between two specified variables*

Description

Estimate the total causal effect between two specified variables

Usage

```
estimate_total_effect(
  X,
  lingam_result,
  from_index,
  to_index,
  method = "adaptive_lasso",
  lambda = "BIC",
  init_method = "ols"
)
```

Arguments

X	Original data (matrix or data.frame)
lingam_result	Return value of lingam_direct()
from_index	Cause variable (1-based index or variable name)
to_index	Effect variable (1-based index or variable name)
method	Regression method ("ols", "lasso", "adaptive_lasso", "ridge"). Default is adaptive_lasso
lambda	Lambda selection ("lambda.min", "lambda.1se", "AIC", "BIC", "oracle"). Default is BIC
init_method	Method for estimating the initial weights of adaptive LASSO regression ("ols" or "ridge")

Value

Estimated total causal effect

Examples

```
LiNGAM_sample_1000 <- generate_lingam_sample_6()

model <- LiNGAM_sample_1000$data |>
  lingam_direct(reg_method = "ols")

LiNGAM_sample_1000$data |>
  estimate_total_effect(model, 4, 1)
```

---

estimate_total_effect_parce

Estimate the total causal effect between two variables (ParceLiNGAM)

Description

Analogous to `estimate_total_effect()`, but for `lingam_parce()` results, which may contain `NA` entries in the adjacency matrix.

Usage

```
estimate_total_effect_parce(
  X,
  parce_result,
  from_index,
  to_index,
  method = "adaptive_lasso",
  lambda = "BIC",
  init_method = "ols"
)
```

Arguments

X	Original data (matrix or data.frame)
parce_result	Return value of lingam_parce()
from_index	Cause variable (1-based index or variable name)
to_index	Effect variable (1-based index or variable name)
method	Regression method ("ols", "lasso", "adaptive_lasso", "ridge"). Default is adaptive_lasso
lambda	Lambda selection ("lambda.min", "lambda.1se", "AIC", "BIC", "oracle"). Default is BIC
init_method	Method for estimating the initial weights of adaptive LASSO regression ("ols" or "ridge")

Value

Estimated total causal effect, or NA (with a warning) if from_index is part of an unresolved block (its parents cannot be identified).

Examples

```
confounded <- generate_parce_sample(n = 500, seed = 1)
result <- lingam_parce(confounded$data, reg_method = "ols")

# A well-identified pair returns a numeric estimate
estimate_total_effect_parce(confounded$data, result, from_index = 1, to_index = 5)
```

---

estimate_total_effect_rcd

Estimate the total causal effect between two variables (RCD)

Description

Analogous to estimate_total_effect(), but for lingam_rcd() results, which may contain NA entries in the adjacency matrix.

Usage

```
estimate_total_effect_rcd(
  X,
  rcd_result,
  from_index,
  to_index,
  method = "adaptive_lasso",
  lambda = "BIC",
  init_method = "ols"
)
```

Arguments

X	Original data (matrix or data.frame)
rcd_result	Return value of lingam_rcd()
from_index	Cause variable (1-based index or variable name)
to_index	Effect variable (1-based index or variable name)
method	Regression method ("ols", "lasso", "adaptive_lasso", "ridge"). Default is adaptive_lasso
lambda	Lambda selection ("lambda.min", "lambda.1se", "AIC", "BIC", "oracle"). Default is BIC
init_method	Method for estimating the initial weights of adaptive LASSO regression ("ols" or "ridge")

Value

Estimated total causal effect, or NA (with a warning) if from_index is part of a suspected latent confounder pair (its parents cannot be identified). Also warns (without altering the estimate) if to_index is an ancestor of from_index according to ancestors_list, since that is inconsistent with a from -> to effect.

Examples

```
confounded <- generate_rcd_sample(n = 300, seed = 1)
result <- lingam_rcd(confounded$data)

# A well-identified pair returns a numeric estimate
estimate_total_effect_rcd(confounded$data, result, from_index = 6, to_index = 1)
```

---

estimate_var_total_effect

Estimate a total causal effect in a VAR-LiNGAM model

Description

Estimates the total causal effect from from_index (optionally at lag from_lag) to to_index (at the current time) using the fitted VAR-LiNGAM model. Port of the Python reference estimate_total_effect: the destination variable is regressed on the source variable together with the source's parents (a back-door adjustment), and the source's coefficient is returned.

Usage

```
estimate_var_total_effect(X, result, from_index, to_index, from_lag = 0)
```

Arguments

X	original data (matrix or data frame), rows ordered in time
result	a VARLiNGAMResult from lingam_var()
from_index	source variable (1-based index or variable name)
to_index	destination variable (1-based index or variable name)
from_lag	lag of the source variable (0 = current time, default)

Value

the estimated total effect (scalar)

Examples

```
sample <- generate_varlingam_sample(n = 1000, seed = 42)
model <- lingam_var(sample$data, lags = 1, reg_method = "ols", prune = FALSE)

# total effect of x0 (current) on x2 (current)
estimate_var_total_effect(sample$data, model, from_index = 1, to_index = 3)
```

---

evaluate_model_fit *Evaluate model fit of an estimated causal graph*

Description

Fits the causal graph implied by adjacency_matrix as a structural equation model (SEM) via lavaan::sem() and returns standard SEM fit measures (CFI, RMSEA, AIC/BIC, etc.). This is an R port of the Python lingam.utils.evaluate_model_fit(), which delegates to the Python package semopy; this R version delegates to lavaan instead.

Usage

```
evaluate_model_fit(adjacency_matrix, X, is_ordinal = NULL)
```

Arguments

adjacency_matrix	p x p numeric adjacency matrix (NA allowed for latent confounder pairs), or a lingamr result object (e.g. LingamResult, ParceLingamResult, LiMResult) with an adjacency_matrix element, from which the matrix is extracted automatically
X	numeric matrix or data frame (n_samples x p) with no missing values
is_ordinal	logical or 0/1 vector of length p. TRUE marks a variable as ordinal (categorical), fit with lavaan's WLSMV-based estimator. NULL (default) treats all variables as continuous.

Details

- **Optional dependency**: this function requires the **lavaan** package (listed in `Suggests`, not `Imports`). Install it with `install.packages("lavaan")`.
- **Latent confounders**: an `NA` element `B[i, j]` (as produced by e.g. `lingam_parce()` for a suspected latent confounder between variables i and j) is represented as a residual covariance `xi ~~ xj` in the lavaan model. This is algebraically equivalent to the two-indicator latent common cause (one loading fixed to 1) used by the Python `semopy` implementation, but is expressed with lavaan's standard residual-covariance idiom rather than an explicit latent variable.
- **Numerical values will not match `semopy` exactly**: `lavaan` and `semopy` use different default estimators/options. The fit measures returned are the same statistics, but exact numbers can differ.
- **Convention**: `adjacency_matrix` follows the lingamr convention `B[i, j]` = causal coefficient from variable j to variable i (j -> i).

Value

A one-row data.frame of fit measures: DoF, DoF Baseline, chi2, chi2 p-value, chi2 Baseline, CFI, GFI, AGFI, NFI, TLI, RMSEA, AIC, BIC, LogLik. When `is_ordinal` is used, AIC/BIC/LogLik and some other measures are not defined by the WLSMV estimator and are returned as `NA`.

References

Rosseel, Y. (2012). lavaan: An R Package for Structural Equation Modeling. Journal of Statistical Software, 48(2), 1-36. doi:10.18637/jss.v048.i02

Examples

```
if (requireNamespace("lavaan", quietly = TRUE)) {
  dat <- generate_lingam_sample_6()
  result <- lingam_direct(dat$data, reg_method = "ols")
  evaluate_model_fit(result, dat$data)
}
```

`generate_lim_sample` *Generate sample data for LiM (3 mixed variables)*

Description

Generates a small dataset with a known causal chain of continuous and binary (0/1) discrete variables: `x1 (continuous) -> x2 (discrete) -> x3 (continuous)`. Continuous variables use Laplace-distributed noise (non-Gaussian, as required by LiNGAM-family methods); the discrete variable is drawn from a Bernoulli distribution whose logit is a linear function of its parent.

Usage

```
generate_lim_sample(n = 1000L, seed = NULL)
```

Arguments

n	number of samples (default: 1000)
seed	random seed. If NULL (default), no seed is set and results are not reproducible across calls.

Value

A list with three elements:

- data: data frame with columns x1, x2, x3.
- adjacency_matrix: 3x3 true adjacency matrix, following the lingamr convention (m[to, from], i.e. adjacency_matrix["x2", "x1"] is the coefficient of the x1 -> x2 edge).
- is_continuous: logical vector c(TRUE, FALSE, TRUE) marking which columns of data are continuous.

Examples

```
dat <- generate_lim_sample(n = 500, seed = 1)
result <- lingam_lim(dat$data, is_continuous = dat$is_continuous)
result$adjacency_matrix
```

---

generate_lingam_hard_sample

Generate a challenging sample data for Direct LiNGAM

Description

Generates a dataset with conditions that make causal estimation difficult:

1. High multicollinearity among predictors
2. Moderate sample size relative to variables
3. True coefficients of similar magnitude

Usage

```
generate_lingam_hard_sample(n = 200L, seed = 42L, collinearity = 0.95)
```

Arguments

n	number of samples (default: 200)
seed	random seed (default: 42)
collinearity	strength of multicollinearity (0 to 1, default: 0.95)

Details

These conditions destabilize OLS initial estimates in Adaptive LASSO, making Ridge-initialized Adaptive LASSO preferable.

Value

list(data, true_adjacency)

Examples

```
result <- generate_lingam_hard_sample()
result$true_adjacency
head(result$data)
```

---

generate_lingam_large_sample

Generate large-scale sample data to benchmark Direct LiNGAM scalability

---

Description

Generates a dataset with many variables to demonstrate the computational scalability difference between Direct LiNGAM and ICA-LiNGAM.

Usage

```
generate_lingam_large_sample(
  p = 20L,
  n = 1000L,
  max_parents = 3L,
  coef_min = 0.5,
  coef_max = 1.5,
  seed = 42L,
  noise_dist = "uniform"
)
```

Arguments

p	number of variables (default: 20)
n	number of observations (default: 1000)
max_parents	maximum number of parents per node (default: 3). Controls graph density. Each variable xi (i >= 1) receives between 1 and `min(max_parents, i)` parents drawn from x0, ..., x(i-1).
coef_min	minimum absolute value of edge coefficients (default: 0.5)
coef_max	maximum absolute value of edge coefficients (default: 1.5)

seed	random seed (default: 42)
noise_dist	error term distribution. "uniform" : Uniform(0, 1) - default, non-Gaussian (LiNGAM works well) "gaussian" : Normal(0, 1) - LiNGAM may fail "lognormal" : Log-normal(0, 1) - skewed, non-Gaussian "exponential" : Exponential(1) - skewed, non-Gaussian "t3" : t-distribution (df=3) - heavy tails

Details

Why Direct LiNGAM slows down with large p:

At each of its p steps, Direct LiNGAM evaluates an independence measure between every remaining candidate root and every other residual. The total number of evaluations is:

$$\sum_{k=1}^{p} k(k-1) \approx \frac{p^3}{3}$$

i.e., O(p^3). Each evaluation is itself O(n), giving O(p^3 n) overall. For p = 10 this is about 330 evaluations; for p = 20 about 2,660; for p = 40 about 21,320 — an 8x increase every time p doubles.

Why ICA-LiNGAM scales better:

ICA-LiNGAM applies FastICA once to the whole p x n data matrix. Each FastICA iteration costs O(p^2 n), and the algorithm typically converges in far fewer than p iterations. Additionally, these matrix operations are fully vectorised (BLAS/LAPACK), whereas Direct LiNGAM iterates over pairs in an R loop.

Data-generating process:

Variables are topologically ordered as x0, x1, ..., x(p-1). For each i >= 1, the number of parents is sampled uniformly from 1 to `min(max_parents, i)`, and the parents are drawn without replacement from x0, ..., x(i-1). Edge coefficients are drawn uniformly from [-coef_max, -coef_min] U [coef_min, coef_max]. The resulting adjacency matrix is strictly lower-triangular.

Value

A list with three elements:

- `data`: data.frame with p columns (x0, x1, ..., x(p-1)).
- `true_adjacency`: p x p matrix. `true_adjacency[i, j]` is the structural coefficient of the edge xj -> xi (row = to, col = from). The matrix is strictly lower-triangular because variables are stored in causal order.
- `true_causal_order`: integer vector `0:(p-1)`. Variables are already in topological order, so the true causal order is always 0, 1, ..., p-1.

Examples

```
# Generate 20-variable data and check its sparsity
dataset <- generate_lingam_large_sample(p = 20, n = 500)
dim(dataset$data)                    # 500 x 20
sum(dataset$true_adjacency != 0)     # number of edges
dataset$true_causal_order            # 0, 1, ..., 19
```

```
# As the number of variables grows, Direct LiNGAM's run time increases sharply
t10 <- system.time(lingam_direct(generate_lingam_large_sample(p = 10)$data))
t20 <- system.time(lingam_direct(generate_lingam_large_sample(p = 20)$data))
cat(sprintf("p=10: %.1f sec,  p=20: %.1f sec\n", t10["elapsed"], t20["elapsed"]))
```

`generate_lingam_paradox_data` *Generate Paradoxical Data Where DirectLiNGAM Struggles*

Description

Generates a synthetic dataset designed to favor ICA-LiNGAM (due to standardized scales) while challenging DirectLiNGAM (due to heavy measurement noise on the root variable, which triggers error propagation). The true causal structure is a serial chain: `x0 -> x1 -> x2 -> x3` (each coefficient 0.8).

Usage

```
generate_lingam_paradox_data(n = 2000L, seed = 42L)
```

Arguments

`n`	number of samples (default: 2000)
`seed`	random seed (default: 42)

Details

This function intentionally injects strong measurement error into the root (causal upstream) variable `x0`. This noise corrupts the independence tests performed at the initial step of DirectLiNGAM, frequently causing it to misidentify the root variable and leading to a cascading failure (error propagation) throughout the causal ordering.

On the other hand, the output data is completely standardized using the `scale()` function. This eliminates any differences in scale among the variables, thereby neutralizing the major weakness of ICA-LiNGAM (scale-dependence) and allowing it to perform relatively better.

Because the data are standardized and the root carries measurement error, the coefficients estimated by `lingam_direct()` will not exactly match the 0.8 values stored in `true_adjacency`.

Value

list(data, true_adjacency)

- `data`: a data frame with 4 standardized variables (`x0`, `x1`, `x2`, `x3`); each column has a mean of 0 and a standard deviation of 1.

- `true_adjacency`: the 4x4 true adjacency matrix of the data-generating chain, following the `m[row = to, col = from]` convention and holding the structural coefficients (0.8) on the latent, pre-standardization scale.

Examples

```
# Generate the dataset
paradox <- generate_lingam_paradox_data(n = 1000, seed = 123)

# Verify the dataset
head(paradox$data)
sapply(paradox$data, sd)

# True data-generating structure
paradox$true_adjacency
```

`generate_lingam_sample_10`

Generate 10-variable sample data for Direct LiNGAM

Description

Generates a sample dataset with a known causal structure. The true causal structure is: x3 -> x0 (coef = 3.0) x3 -> x2 (coef = 6.0) x3 -> x9 (coef = 7.0) x0 -> x1 (coef = 3.0) x0 -> x5 (coef = 4.0) x0 -> x4 (coef = 8.0) x0 -> x7 (coef = 3.0) x2 -> x1 (coef = 2.0) x2 -> x4 (coef = -1.0) x2 -> x8 (coef = 0.5) x1 -> x6 (coef = 2.0) x5 -> x8 (coef = 2.0) x4 -> x7 (coef = 1.5) x6 -> x9 (coef = 1.0)

Usage

```
generate_lingam_sample_10(n = 1000L, seed = 42L, noise_dist = "uniform")
```

Arguments

`n`	number of samples (default: 1000)
`seed`	random seed (default: 42)
`noise_dist`	error term distribution "uniform" : Uniform(0, 1) - default, non-Gaussian (LiNGAM works well) "gaussian" : Normal(0, 1) - LiNGAM may fail "lognormal" : Log-normal(0, 1) - skewed, non-Gaussian "exponential" : Exponential(1) - skewed, non-Gaussian "t3" : t-distribution (df=3) - heavy tails

Value

list(data, true_adjacency)

Examples

```
# Non-Gaussian (LiNGAM works well)
X_nongauss <- generate_lingam_sample_10(noise_dist = "uniform")
result <- lingam_direct(X_nongauss$data, reg_method = "ols")
result$causal_order

# Gaussian (LiNGAM may fail)
X_gauss <- generate_lingam_sample_10(noise_dist = "gaussian")
result <- lingam_direct(X_gauss$data, reg_method = "ols")
result$causal_order
```

`generate_lingam_sample_6`

Generate sample data for Direct LiNGAM (6 variables)

Description

Generates a sample dataset with a known causal structure. The true causal structure is: x3 -> x0 (coef = 3.0) x3 -> x2 (coef = 6.0) x0 -> x1 (coef = 3.0) x2 -> x1 (coef = 2.0) x0 -> x5 (coef = 4.0) x0 -> x4 (coef = 8.0) x2 -> x4 (coef = -1.0)

Usage

```
generate_lingam_sample_6(n = 1000L, seed = 42L, noise_dist = "uniform")
```

Arguments

`n`	number of samples (default: 1000)
`seed`	random seed (default: 42)
`noise_dist`	error term distribution "uniform" : Uniform(0, 1) - default, non-Gaussian (LiNGAM works well) "gaussian" : Normal(0, 1) - LiNGAM may fail "lognormal" : Log-normal(0, 1) - skewed, non-Gaussian "exponential" : Exponential(1) - skewed, non-Gaussian "t3" : t-distribution (df=3) - heavy tails

Value

list(data, true_adjacency)

Examples

```
# Non-Gaussian (LiNGAM works well)
X_nongauss <- generate_lingam_sample_6(noise_dist = "uniform")
result <- lingam_direct(X_nongauss$data, reg_method = "ols")
result$causal_order

# Gaussian (LiNGAM may fail)
X_gauss <- generate_lingam_sample_6(noise_dist = "gaussian")
```

```
result <- lingam_direct(X_gauss$data, reg_method = "ols")
result$causal_order
```

`generate_multi_group_sample` *Generate sample data for Multi-Group Direct LiNGAM (2 groups, 6 variables)*

Description

Generates two datasets that share the same causal structure as `generate_lingam_sample_6()` (`x3 -> x0, x3 -> x2, x0 -> x1, x2 -> x1, x0 -> x4, x2 -> x4, x0 -> x5`) but with different structural coefficients per group, following the multi-dataset tutorial's setup.

Usage

```
generate_multi_group_sample(n = c(1000, 1000), seed = 42L)
```

Arguments

`n`	Numeric vector of length 2: sample size per group (default `c(1000, 1000)`).
`seed`	Random seed (default 42). Group 2 uses an internally offset seed so the two groups are independently drawn.

Value

A list with:

- `data_list`: named list (`group1`, `group2`) of data frames, each with columns `x0`..`x5`.
- `adjacency_matrices`: named list of the true adjacency matrix per group (`m[to, from] = coef`, same convention as `lingam_direct()`).
- `causal_order`: the true causal order shared by both groups (1-based indices into `x0`..`x5`).

Examples

```
mg <- generate_multi_group_sample()
lapply(mg$data_list, head)
mg$adjacency_matrices$group1
```

generate_parce_sample *Generate sample data with a latent confounder (for BottomUpParceLiNGAM)*

Description

Generates the 7-variable model used in the ParceLiNGAM tutorial, where `x6` is an unobserved (latent) common cause of `x2` and `x3`. Only `x0`-`x5` are returned as observed data; `x6` is not included.

Usage

```
generate_parce_sample(n = 1000L, seed = NULL)
```

Arguments

`n`	number of samples (default: 1000)
`seed`	random seed (default: NULL, i.e. do not reset the RNG state)

Details

The data-generating process (all error terms are `Uniform(0, 1)`):

```
x6 (latent) ~ Uniform(0, 1)
x3 = 2.0 * x6 + e
x2 = 2.0 * x6 + e
x0 = 0.5 * x3 + e
x1 = 0.5 * x0 + 0.5 * x2 + e
x5 = 0.5 * x0 + e
x4 = 0.5 * x0 - 0.5 * x2 + e
```

Value

list with three elements:

- `data`: data.frame of the 6 observed variables (`x0`-`x5`).
- `adjacency_matrix`: the true 6x6 adjacency matrix among the observed variables, following the `m[to, from]` convention. The `x2`-`x3` entries (which share the latent confounder `x6` and have no direct edge between them) are `NA`, matching the convention used by `lingam_parce()`.
- `confounded_pair`: 1-based column positions of `x2` and `x3` (the pair sharing the latent confounder).

Examples

```
confounded <- generate_parce_sample(n = 500, seed = 42)
head(confounded$data)
confounded$adjacency_matrix
confounded$confounded_pair
```

`generate_rcd_sample` *Generate sample data with a latent confounder (for RCD)*

Description

Generates the 7-variable model used in the RCD tutorial, where `x6` is an unobserved (latent) common cause of `x2` and `x4`. Only `x0`-`x5` are returned as observed data; `x6` is not included.

Usage

```
generate_rcd_sample(n = 300L, seed = NULL)
```

Arguments

`n`	number of samples (default: 300)
`seed`	random seed (default: NULL, i.e. do not reset the RNG state)

Details

The data-generating process (all error terms `e()` are super-Gaussian, `rnorm(n, 0, 0.5)^3`):

```
x5 ~ e();  x6 (latent) ~ e()
x1 = 0.6 * x5 + e()
x3 = 0.5 * x5 + e()
x0 = 1.0 * x1 + 1.0 * x3 + e()
x2 = 0.8 * x0 - 0.6 * x6 + e()
x4 = 1.0 * x0 - 0.5 * x6 + e()
```

Value

list with four elements:

- `data`: data.frame of the 6 observed variables (`x0`-`x5`).
- `adjacency_matrix`: the true 6x6 adjacency matrix among the observed variables, following the `m[to, from]` convention. The `x2`-`x4` entries (which share the latent confounder `x6` and have no direct edge between them) are `NA`, matching the convention used by `lingam_rcd()`.
- `ancestors_list`: the true ancestor sets (1-based column positions), usable as a test oracle for `lingam_rcd()`'s `ancestors_list`.
- `confounded_pair`: 1-based column positions of `x2` and `x4` (the pair sharing the latent confounder).

Examples

```
confounded <- generate_rcd_sample(n = 300, seed = 1)
head(confounded$data)
confounded$adjacency_matrix
confounded$ancestors_list
confounded$confounded_pair
```

generate_varlingam_sample

Generate sample data from a VAR-LiNGAM model

Description

Generates a 3-variable time series following a VAR-LiNGAM model with a strictly acyclic instantaneous structure B0, a lag-1 coefficient matrix M1, and non-Gaussian (uniform) errors.

Usage

```
generate_varlingam_sample(n = 1000, seed = NULL)
```

Arguments

n	number of time points to return (after burn-in)
seed	random seed (NULL allowed)

Value

list with `data` (data frame, n x 3), `true_B0` (instantaneous matrix), and `true_M1` (lag-1 coefficient matrix)

Examples

```
sample <- generate_varlingam_sample(n = 500, seed = 1)
head(sample$data)
```

get_adjacency_matrix_summary

Create an adjacency matrix of representative causal-effect values from bootstrap results

Description

Create an adjacency matrix of representative causal-effect values from bootstrap results

Usage

```
get_adjacency_matrix_summary(
  result,
  stat = "median",
  min_causal_effect = NULL,
  min_probability = NULL,
  labels = NULL
)
```

Arguments

`result`	BootstrapResult object
`stat`	Representative statistic ("mean" or "median")
`min_causal_effect`	Minimum threshold for the causal effect (values at or below this are treated as zero) (NULL = 0)
`min_probability`	Edges below this probability are set to zero (NULL = 0)
`labels`	Vector of variable names (NULL allowed)

Value

Adjacency matrix (n_features x n_features). **Rule:** `B[i, j]` **is the causal coefficient from variable j to variable i (j -> i).** Same rule as the `adjacency_matrix` of `lingam_direct()`.

Examples

```
LiNGAM_sample_1000 <- generate_lingam_sample_6()

bs_model <- lingam_direct_bootstrap(LiNGAM_sample_1000$data,
  n_sampling = 30L, reg_method = "ols", seed = 42
)
get_adjacency_matrix_summary(bs_model)
```

`get_causal_direction_counts` *Get counts, proportions, and causal effects of causal directions*

Description

Get counts, proportions, and causal effects of causal directions

Usage

```
get_causal_direction_counts(
  result,
  n_directions = NULL,
  min_causal_effect = NULL,
  split_by_causal_effect_sign = FALSE,
  labels = NULL
)
```

Arguments

result	BootstrapResult object
n_directions	How many of the top entries to return (NULL = all)
min_causal_effect	Minimum threshold for the causal effect (NULL = 0)
split_by_causal_effect_sign	Whether to split by the sign of the causal effect
labels	Vector of variable names (NULL allowed; if provided, adds from_name and to_name columns)

Value

A data frame containing the following columns:

- from, to: 1-based indices of the causal (from) and effect (to) variables.
- count: Number of bootstrap samples in which this specific causal direction was identified.
- proportion: The frequency of the direction's occurrence (count / n_sampling), representing its bootstrap probability.
- mean_effect: The average value of the estimated causal effects across samples where this direction was identified.
- median_effect: The median value of the estimated causal effects, providing a robust estimate of the effect size.
- sd_effect: The standard deviation of the causal effect estimates, indicating the stability of the effect size.
- ci_lower, ci_upper: The lower (2.5%) and upper (97.5%) bounds of the bootstrap confidence interval for the causal effect.
- sign (optional): The sign of the causal effect (1 for positive, -1 for negative), included if split_by_causal_effect_sign = TRUE.
- from_name, to_name (optional): Character labels for the variables, included if labels were provided.

Examples

```
LiNGAM_sample_1000 <- generate_lingam_sample_6()

bs_model <- lingam_direct_bootstrap(LiNGAM_sample_1000$data,
  n_sampling = 30L, reg_method = "ols", seed = 42
)

get_causal_direction_counts(bs_model, labels = names(LiNGAM_sample_1000$data))
```

get_causal_order_stability *Evaluate the stability of the causal order from bootstrap*

Description

Aggregates the causal order (causal_order) estimated in each bootstrap sample and quantifies how stable the order is. Returns the rank distribution of each variable, the precedence probabilities for variable pairs, and an overall stability score.

Usage

```
get_causal_order_stability(result, labels = NULL)
```

Arguments

result	A BootstrapResult object (run with the current version)
labels	A vector of variable names (if NULL, x0, x1, ... are generated automatically)

Value

A list of class causal_order_stability, containing:

- rank_summary: A summary of the rank of each variable (variable, mean_rank, sd_rank, median_rank, mode_rank). Sorted in ascending order of mean_rank (from upstream). A rank of 1 is the most upstream.
- precedence_matrix: A precedence probability matrix. P[i, j] is the proportion of bootstrap samples in which variable i was located upstream of (before) variable j.
- stability_score: An overall stability score, from 0 (random order) to 1 (order agrees across all samples). The closer the precedence probability of each variable pair is to 0/1, the higher the score.
- n_sampling: The number of bootstrap samples.

Examples

```
dat <- generate_lingam_sample_6()
bs <- lingam_direct_bootstrap(dat$data, n_sampling = 30L, reg_method = "ols", seed = 42)
get_causal_order_stability(bs, labels = names(dat$data))
```

`get_directed_acyclic_graph_counts` *Get DAG counts*

Description

Get DAG counts

Usage

```
get_directed_acyclic_graph_counts(
  result,
  n_dags = NULL,
  min_causal_effect = NULL,
  split_by_causal_effect_sign = FALSE
)
```

Arguments

`result`	BootstrapResult object
`n_dags`	How many of the top entries to return (NULL = all)
`min_causal_effect`	Minimum threshold for the causal effect (NULL = 0)
`split_by_causal_effect_sign`	Whether to split by the sign of the causal effect

Value

list(dag = list of data.frames, count = integer vector)

Examples

```
LiNGAM_sample_1000 <- generate_lingam_sample_6()

bs_model <- lingam_direct_bootstrap(LiNGAM_sample_1000$data,
  n_sampling = 30L, reg_method = "ols", seed = 42
)

get_directed_acyclic_graph_counts(bs_model)
```

`get_error_independence_p_values` *Compute p-values for the independence test of the errors*

Description

Compute p-values for the independence test of the errors

Usage

```
get_error_independence_p_values(X, lingam_result, method = "spearman")
```

Arguments

`X`	original data (matrix or data.frame)
`lingam_result`	return value of lingam_direct()
`method`	type of correlation coefficient ("spearman", "pearson", "kendall"). "kendall" uses the O(n^2)-per-pair algorithm in `stats::cor.test()`; for large n (beyond 5000) this warns, and "spearman" is a much faster alternative with similar rank-based semantics.

Value

matrix of p-values (n_features x n_features)

Examples

```
# Load the sample data
LiNGAM_sample_1000 <- generate_lingam_sample_6()

# Run Direct LiNGAM
result <- LiNGAM_sample_1000$data |>
  lingam_direct(reg_method = "ols")

# Compute p-values (default: Spearman)
p_vals <- get_error_independence_p_values(LiNGAM_sample_1000$data, result)
round(p_vals, 3)

# Compute with Kendall
p_vals_k <- get_error_independence_p_values(LiNGAM_sample_1000$data, result, method = "kendall")
round(p_vals_k, 3)
```

get_error_independence_p_values_parce

Compute p-values for the independence of ParceLiNGAM residuals (HSIC-based)

Description

Analogous to `get_error_independence_p_values()`, but for `lingam_parce()` results. Uses the HSIC gamma-approximation test (`hsic_test_gamma()`) rather than a correlation test, and returns `NA` for any pair involving a variable whose row or column in the adjacency matrix contains `NA` (residuals cannot be computed for those variables).

Usage

```
get_error_independence_p_values_parce(X, parce_result)
```

Arguments

`X`	Original data (matrix or data.frame)
`parce_result`	Return value of `lingam_parce()`

Value

matrix of p-values (n_features x n_features)

Examples

```
confounded <- generate_parce_sample(n = 500, seed = 1)
result <- lingam_parce(confounded$data, reg_method = "ols")
round(get_error_independence_p_values_parce(confounded$data, result), 3)
```

get_error_independence_p_values_rcd

Compute p-values for the independence of RCD residuals (HSIC-based)

Description

Analogous to `get_error_independence_p_values_parce()`, but for `lingam_rcd()` results. Returns `NA` for any pair involving a variable whose row or column in the adjacency matrix contains `NA` (residuals cannot be computed for those variables).

Usage

```
get_error_independence_p_values_rcd(X, rcd_result)
```

Arguments

X	Original data (matrix or data.frame)
rcd_result	Return value of lingam_rcd()

Value

matrix of p-values (n_features x n_features)

Examples

```
confounded <- generate_rcd_sample(n = 300, seed = 1)
result <- lingam_rcd(confounded$data)
round(get_error_independence_p_values_rcd(confounded$data, result), 3)
```

---

get_group_result *Extract a single group's result from a MultiGroupLingamResult*

Description

Returns the adjacency matrix and (shared) causal order of one group as a plain LingamResult, so that the existing single-group functions (estimate_total_effect(), estimate_all_total_effects(), get_error_independence_p_values(), plot_adjacency(), autoplot(), tidy()) can be applied to it directly.

Usage

```
get_group_result(x, group)
```

Arguments

x	A MultiGroupLingamResult, as returned by lingam_multi_group().
group	Group name (character) or 1-based group index (integer).

Value

A LingamResult object (list) with adjacency_matrix and causal_order, identical in shape to the return value of lingam_direct().

Examples

```
mg <- generate_multi_group_sample()
res <- lingam_multi_group(mg$data_list, reg_method = "ols")
g1 <- get_group_result(res, 1)
class(g1)
```

get_paths *Get all paths between two specified variables and their bootstrap probabilities*

Description

Get all paths between two specified variables and their bootstrap probabilities

Usage

```
get_paths(result, from_index, to_index, min_causal_effect = NULL)
```

Arguments

result	BootstrapResult object
from_index	Start index (1-based)
to_index	End index (1-based)
min_causal_effect	Minimum threshold for the causal effect (NULL = 0)

Value

data.frame (path, effect, probability)

Examples

```
LiNGAM_sample_1000 <- generate_lingam_sample_6()

bs_model <- lingam_direct_bootstrap(LiNGAM_sample_1000$data,
  n_sampling = 30L, reg_method = "ols", seed = 42
)
get_paths(bs_model, 1, 6)
```

get_probabilities *Get bootstrap probabilities*

Description

Get bootstrap probabilities

Usage

```
get_probabilities(result, min_causal_effect = NULL)
```

Arguments

result	BootstrapResult object
min_causal_effect	Minimum threshold for the causal effect (NULL = 0)

Value

Probability matrix (n_features x n_features)

Examples

```
LiNGAM_sample_1000 <- generate_lingam_sample_6()

bs_model <- lingam_direct_bootstrap(LiNGAM_sample_1000$data,
  n_sampling = 30L, reg_method = "ols", seed = 42
)

get_probabilities(bs_model)
```

---

get_total_causal_effects

Get a list of total causal effects

Description

Get a list of total causal effects

Usage

```
get_total_causal_effects(result, min_causal_effect = NULL)
```

Arguments

result	BootstrapResult object
min_causal_effect	Minimum threshold for the causal effect (NULL = 0)

Value

data.frame (from, to, effect, probability)

Examples

```
LiNGAM_sample_1000 <- generate_lingam_sample_6()

bs_model <- lingam_direct_bootstrap(LiNGAM_sample_1000$data,
  n_sampling = 30L, reg_method = "ols", seed = 42
)

get_total_causal_effects(bs_model)
```

---

get_var_paths *Enumerate bootstrap paths between two variables in a VAR-LiNGAM model*

Description

Builds the time-expanded graph for every bootstrap sample and enumerates all directed paths from the source (at `from_lag`) to the destination (at `to_lag`), reporting each path's bootstrap probability and median effect. Port of the Python reference `VARBootstrapResult.get_paths`.

Usage

```
get_var_paths(
  result,
  from_index,
  to_index,
  from_lag = 0,
  to_lag = 0,
  min_causal_effect = NULL
)
```

Arguments

`result`	a VARBootstrapResult object
`from_index`	source variable (1-based)
`to_index`	destination variable (1-based)
`from_lag`	lag of the source (default 0)
`to_lag`	lag of the destination (default 0); must satisfy `to_lag <= from_lag`
`min_causal_effect`	minimum |effect| threshold (NULL = 0)

Details

Node indices in the returned `path` are 1-based positions in the time-expanded graph: column j of block L (lag L) corresponds to index `n_features * L + j`.

Value

a data frame (path, effect, probability), one row per distinct path

Examples

```
s <- generate_varlingam_sample(n = 500, seed = 42)
bs <- lingam_var_bootstrap(s$data,
  n_sampling = 10L, criterion = NULL,
  reg_method = "ols", prune = FALSE, seed = 1, verbose = FALSE
)
get_var_paths(bs, from_index = 1, to_index = 3)
```

---

get_var_probabilities *Bootstrap probabilities for a VAR-LiNGAM model*

Description

Returns, for each entry of the joined adjacency matrix, the fraction of bootstrap samples in which that edge exceeded `min_causal_effect`.

Usage

```
get_var_probabilities(result, min_causal_effect = NULL)
```

Arguments

`result`	a VARBootstrapResult object
`min_causal_effect`	minimum |effect| threshold (NULL = 0)

Value

probability matrix (n_features x n_features*(1 + lags)). Columns 1..n_features are the instantaneous block; the next n_features are lag 1; etc. `P[i, j]` is the probability of the edge j -> i.

Examples

```
s <- generate_varlingam_sample(n = 500, seed = 42)
bs <- lingam_var_bootstrap(s$data,
  n_sampling = 10L, criterion = NULL,
  reg_method = "ols", prune = FALSE, seed = 1, verbose = FALSE
)
get_var_probabilities(bs)
```

glance.LiMResult *Get a one-row summary of a LiMResult*

Description

Like glance.LingamResult(), with an additional n_discrete column giving the number of discrete variables in the model.

Usage

```
## S3 method for class 'LiMResult'
glance(x, ...)
```

Arguments

x	The return value of lingam_lim() (a LiMResult object)
...	Unused

Value

A one-row data.frame(n_variables, n_edges, n_discrete, causal_order)

Examples

```
set.seed(1)
dat <- generate_lim_sample(n = 300)
model <- lingam_lim(dat$data, is_continuous = dat$is_continuous)
glance(model)
```

glance.LingamResult *Get a one-row summary of a LingamResult*

Description

Summarizes the entire model in a single row. The data X is not required because no residuals are computed. If residual-based diagnostics are needed, use summary_lingam() instead.

Usage

```
## S3 method for class 'LingamResult'
glance(x, ...)
```

Arguments

x	The return value of lingam_direct() (a LingamResult object)
...	Unused

Value

A one-row data.frame(n_variables, n_edges, causal_order)

Examples

```
dat <- generate_lingam_sample_6()
model <- lingam_direct(dat$data, reg_method = "ols")
glance(model)
```

`glance.MultiGroupLingamResult` *Get a one-row summary of a MultiGroupLingamResult*

Description

Summarizes the joint model in a single row. The causal order is shared across groups; per-group edge counts are available via `glance(get_group_result(x, i))`.

Usage

```
## S3 method for class 'MultiGroupLingamResult'
glance(x, ...)
```

Arguments

`x`	The return value of `lingam_multi_group()` (a `MultiGroupLingamResult` object)
`...`	Unused

Value

A one-row data.frame(n_groups, n_variables, causal_order)

Examples

```
mg <- generate_multi_group_sample()
model <- lingam_multi_group(mg$data_list, reg_method = "ols")
glance(model)
```

glance.ParceLingamResult *Get a one-row summary of a ParceLingamResult*

Description

Like `glance.LingamResult()`. `n_edges` counts non-`NA` edges only, and `n_na_entries` counts the adjacency-matrix entries left `NA` (unresolved order / suspected latent confounding). Unresolved blocks in the causal order are shown in parentheses, as in the print method.

Usage

```
## S3 method for class 'ParceLingamResult'
glance(x, ...)
```

Arguments

`x`	The return value of `lingam_parce()` (a `ParceLingamResult` object)
`...`	Unused

Value

A one-row data.frame(n_variables, n_edges, n_na_entries, causal_order)

Examples

```
dat <- generate_parce_sample(n = 500, seed = 42)
model <- lingam_parce(dat$data)
glance(model)
```

glance.RCDResult *Get a one-row summary of an RCDResult*

Description

Like `glance.LingamResult()`, but without a causal order (RCD does not estimate one). `n_edges` counts non-`NA` edges only, and `n_confounded_pairs` counts the variable pairs whose adjacency-matrix entries are `NA` (suspected shared latent confounder).

Usage

```
## S3 method for class 'RCDResult'
glance(x, ...)
```

Arguments

x	The return value of lingam_rcd() (an RCDResult object)
...	Unused

Value

A one-row data.frame(n_variables, n_edges, n_confounded_pairs)

Examples

```
confounded <- generate_rcd_sample(n = 300, seed = 1)
model <- lingam_rcd(confounded$data)
glance(model)
```

---

lingam_direct *Direct LiNGAM*

Description

Direct LiNGAM

Usage

```
lingam_direct(
  X,
  prior_knowledge = NULL,
  apply_prior_knowledge_softly = FALSE,
  measure = "pwling",
  reg_method = "adaptive_lasso",
  lambda = "BIC",
  init_method = "ols"
)
```

Arguments

X	Numeric matrix (n_samples x n_features), data frame or matrix
prior_knowledge	Prior knowledge matrix (n_features x n_features) or NULL. 0: no directed path from x_i to x_j 1: directed path from x_i to x_j -1: unknown
apply_prior_knowledge_softly	Whether to apply prior knowledge softly (logical)
measure	Independence evaluation measure ("pwling" or "kernel")
reg_method	Regression method for adjacency matrix estimation. "ols": ordinary least squares, "lasso": LASSO regression, "adaptive_lasso": adaptive LASSO regression (default), "ridge": Ridge regression (robust to multicollinearity; does not perform sparse estimation).

lambda	LASSO penalty (lambda) selection. "lambda.min" : minimum CV prediction error, prioritizes prediction accuracy. "lambda.1se" : CV 1SE rule, robust and less prone to overfitting. "AIC": minimum AIC. Fast. "BIC": minimum BIC. Fast, sparsest. Default. "oracle" : adaptive LASSO regression only. Selects a lambda that guarantees the oracle property. Fast.
init_method	Method for estimating the initial weights of adaptive LASSO regression. "ols": ordinary least squares (default), "ridge": Ridge regression. Ridge regression is recommended when multicollinearity is suspected.

Value

A LingamResult object (list) containing the following elements:

- adjacency_matrix: adjacency matrix B (n_features x n_features). **Convention: B[i, j] is the causal coefficient from variable j to variable i (j -> i).** Zero elements indicate no causal relationship.
- causal_order: estimated causal order (integer vector of 1-based indices). Earlier elements are more upstream (closer to exogenous variables).

Examples

```
LiNGAM_sample_1000 <- generate_lingam_sample_6()

# OLS (no additional packages required)
result <- lingam_direct(LiNGAM_sample_1000$data, reg_method = "ols")
round(result$adjacency_matrix, 3)


# LASSO (requires glmnet)
result_lasso <- lingam_direct(LiNGAM_sample_1000$data)
round(result_lasso$adjacency_matrix, 3)
```

---

lingam_direct_bootstrap

Bootstrap for Direct LiNGAM

Description

Bootstrap for Direct LiNGAM

Usage

```
lingam_direct_bootstrap(
  X,
  n_sampling,
  prior_knowledge = NULL,
  apply_prior_knowledge_softly = FALSE,
```

```
  measure = "pwling",
  reg_method = "adaptive_lasso",
  lambda = "BIC",
  init_method = "ols",
  seed = NULL,
  verbose = TRUE,
  parallel = FALSE,
  n_cores = NULL,
  compute_total_effects = TRUE
)
```

Arguments

X	Numeric matrix (n_samples x n_features)
n_sampling	Number of bootstrap iterations
prior_knowledge	Prior knowledge matrix (NULL allowed)
apply_prior_knowledge_softly	Apply prior knowledge softly (logical)
measure	Independence measure ("pwling" or "kernel")
reg_method	Regression method ("ols", "lasso", "adaptive_lasso", "ridge")
lambda	Lambda selection ("lambda.min", "lambda.1se", "AIC", "BIC","oracle")
init_method	Method for estimating the initial weights of adaptive LASSO regression ("ols" or "ridge"). Same as the argument of the same name in `lingam_direct()`.
seed	Random seed (NULL allowed)
verbose	Whether to display progress (logical)
parallel	Whether to use parallel processing (logical). When `TRUE`, each bootstrap iteration is distributed across multiple cores.
n_cores	Number of cores to use (integer, NULL allowed). When `NULL`, the number of cores is limited to a maximum of 2 for safety. Ignored when `parallel = FALSE`.
compute_total_effects	Whether to also estimate total causal effects for every variable pair on each bootstrap iteration (logical, default `TRUE`). For the lasso-family regression methods this roughly doubles iteration cost (an additional regression per downstream variable beyond the adjacency-matrix fit). Set to `FALSE` to skip it when only edge/order stability is needed (`get_probabilities()`, `get_causal_direction_counts()`, `get_directed_acyclic_graph_counts()`, `get_causal_order_stability()`); in that case `get_total_causal_effects()` errors if called on the result.

Details

When `parallel = TRUE` is specified, iterations are distributed across a socket cluster created by `parallel::makePSOCKcluster()`. The cluster is always released via `on.exit()`, whether the process finishes normally or an error occurs.

On iteration failures: each bootstrap iteration is run inside a `tryCatch()`. If an iteration errors (e.g. a resample produces near-singular columns), a warning identifying the failed iteration is issued and that iteration is excluded from the result instead of aborting the entire run. The returned `BootstrapResult` reflects however many iterations actually succeeded; an error is raised only if every iteration fails.

On reproducibility: During parallel execution, L'Ecuyer parallel random number streams via `parallel::clusterSetRNGStream()` are used. Results are reproducible given the same `seed` and same `n_cores`, but they do not numerically match the results of sequential execution (`parallel = FALSE`). If you need results that exactly match the sequential version, use `parallel = FALSE`.

Value

BootstrapResult (list)

Examples

```
LiNGAM_sample_1000 <- generate_lingam_sample_6()

# Fast example with OLS
bs <- lingam_direct_bootstrap(LiNGAM_sample_1000$data,
  n_sampling = 10L,
  reg_method = "ols",
  seed = 42
)
get_probabilities(bs)


# With LASSO (requires glmnet)
bs_lasso <- lingam_direct_bootstrap(LiNGAM_sample_1000$data,
  n_sampling = 30L,
  seed = 42
)

# Parallel execution on 2 cores
bs_par <- lingam_direct_bootstrap(LiNGAM_sample_1000$data,
  n_sampling = 30L,
  seed = 42,
  parallel = TRUE,
  n_cores = 2L
)
```

---

lingam_high_dim *High-Dimensional Direct LiNGAM*

Description

A variant of Direct LiNGAM for high-dimensional data (large p, or p > n). Causal order search is based on moment statistics of non-Gaussianity rather than pairwise independence measures, which is considerably faster for many variables. Unlike `lingam_direct()`, the algorithm is deterministic (no random restarts).

Usage

```
lingam_high_dim(X, J = 3L, K = 4L, alpha = 0.5, estimate_adj_mat = TRUE)
```

Arguments

`X`	Numeric matrix (n_samples x n_features), data frame or matrix
`J`	Assumed largest in-degree (single integer, must be >= 3)
`K`	Degree of the moment used to measure non-Gaussianity (single integer, must be >= 1)
`alpha`	Cutoff coefficient for pruning away false parents (single numeric value in `[0, 1]`)
`estimate_adj_mat`	Whether to estimate the adjacency matrix (single logical value). If `FALSE`, causal-order search still runs but `adjacency_matrix` is returned as an NA-filled matrix (not `NULL`, so downstream S3 methods keep working).

Details

When `n_samples` <= `n_features`, the adjacency matrix cannot be estimated with the usual BIC-based Adaptive LASSO, so this function falls back to a cross-validated LASSO (`glmnet::cv.glmnet`) after emitting a warning. The upstream Python implementation uses `LassoLarsCV` for this fallback; `cv.glmnet` follows the same cross-validation design but is not numerically identical (different solver: coordinate descent vs. LARS).

The pruning statistic used during causal-order search (Wang & Drton 2020) is the minimum of a conditional non-Gaussianity statistic over every size-appropriate conditioning subset. The upstream Python implementation (cdt15/lingam) has a `return` statement mis-indented inside the loop over conditioning subsets, causing it to only ever evaluate the first subset. This R implementation intentionally does not replicate that bug and evaluates every subset, so `causal_order` and `adjacency_matrix` are not numerically identical to the upstream Python package.

Value

A `LingamResult` object (list), the same class returned by `lingam_direct()`, containing:

- `adjacency_matrix`: adjacency matrix B (n_features x n_features). **Convention: `B[i, j]` is the causal coefficient from variable j to variable i (j -> i).** Zero elements indicate no causal relationship. All-`NA` when `estimate_adj_mat = FALSE`.
- `causal_order`: estimated causal order (integer vector of 1-based indices). Earlier elements are more upstream.

References

Wang, Y. S. and Drton, M. (2020). High-dimensional causal discovery under non-Gaussianity. Biometrika, 107(1), 41-59.

Examples

```
sample <- generate_lingam_sample_6(n = 300, seed = 1)
result <- lingam_high_dim(sample$data)
result$causal_order
round(result$adjacency_matrix, 3)


if (requireNamespace("glmnet", quietly = TRUE)) {
  # n <= p: falls back to cross-validated LASSO with a warning
  wide_sample <- generate_lingam_large_sample(p = 30, n = 25, seed = 1)
  wide_result <- lingam_high_dim(wide_sample$data)
  wide_result$causal_order
}
```

`lingam_lim` *LiM: LiNGAM for Mixed Data*

Description

Estimates a causal structure from data containing a mixture of continuous and binary (0/1) discrete variables, following Zeng et al. (2022). The method combines a NOTEARS-style continuous optimization (global phase) with a combinatorial local search over edge directions, pruning, and edge addition.

Usage

```
lingam_lim(
  X,
  is_continuous,
  lambda1 = 0.1,
  max_iter = 150L,
  h_tol = 1e-08,
  rho_max = 1e+16,
  w_threshold = 0.1,
  only_global = FALSE
)
```

Arguments

`X`	Numeric matrix (n_samples x n_features) or data frame
`is_continuous`	Logical vector of length `ncol(X)`. `TRUE` marks a continuous variable, `FALSE` marks a discrete (binary 0/1) variable.

lambda1	L1 penalty parameter (default: 0.1)
max_iter	Maximum number of dual ascent (outer loop) steps (default: 150)
h_tol	Tolerance for the acyclicity constraint h(W) (default: 1e-8)
rho_max	Maximum value of the augmented-Lagrangian penalty rho (default: 1e16)
w_threshold	Edges with |weight| below this value are dropped after the global optimization phase (default: 0.1)
only_global	If TRUE, skip the combinatorial local search phase and return the thresholded global-optimization result directly (default: FALSE)

Details

Only binary (0/1) discrete variables are supported; count/Poisson-type discrete variables (the Python source's loss_type = "poisson" path) are not implemented.

The Python implementation's adjacency_matrix_ uses the opposite convention (W[i, j] = i -> j). This R implementation transposes the internal result so that adjacency_matrix follows the lingamr convention (B[i, j] = j -> i), consistent with lingam_direct().

In the local search phase, edges that are reversed or newly added are assigned a weight of exactly 1 rather than a re-estimated coefficient (this matches the original Python implementation). As a result, non-zero entries of adjacency_matrix are a mix of global-phase estimated coefficients and local-phase placeholder weights of 1.

The local search's BIC score for discrete variables with parents uses R's glm(family = binomial()), an unregularized maximum-likelihood fit. The Python implementation uses scikit-learn's LogisticRegression, which applies L2 regularization by default; consequently, numeric results will not exactly match the Python implementation.

Value

A LiMResult object (list) containing the following elements:

- adjacency_matrix: adjacency matrix B (n_features x n_features). **Convention: B[i, j] is the causal coefficient from variable j to variable i (j -> i)**, i.e. the same convention as lingam_direct(). Zero elements indicate no causal relationship.
- causal_order: estimated causal order (integer vector of 1-based indices), derived from adjacency_matrix via a topological sort. NA (with a warning) if the estimated matrix is not acyclic.
- is_continuous: the input is_continuous vector, stored for reference.

References

Zeng Y, Shimizu S, Matsui H, Sun F. Causal discovery for linear mixed data. In: Proceedings of the First Conference on Causal Learning and Reasoning (CLeaR 2022). PMLR 177, pp. 994-1009, 2022.

Examples

```
# Reproducibility requires set.seed(), since the optimization starts from
# a random initial point.
set.seed(1)
dat <- generate_lim_sample(n = 300)
result <- lingam_lim(dat$data, is_continuous = dat$is_continuous)
print(result)
```

`lingam_multi_group` *Multi-Group Direct LiNGAM*

Description

Jointly estimates a Direct LiNGAM model from multiple datasets ("groups") that are assumed to share a common causal order but may have different structural coefficients (Shimizu 2012).

Usage

```
lingam_multi_group(
  X_list,
  prior_knowledge = NULL,
  apply_prior_knowledge_softly = FALSE,
  reg_method = "adaptive_lasso",
  lambda = "BIC",
  init_method = "ols"
)
```

Arguments

`X_list`	A list of numeric matrices or data frames (length >= 2), one per group. Each element must have `n_d` rows (sample size may differ by group) and the same number of columns `p` across all groups.
`prior_knowledge`	Prior knowledge matrix (n_features x n_features) or NULL. Applied identically to every group. Same encoding as `lingam_direct()`: 0: no directed path from x_i to x_j 1: directed path from x_i to x_j -1: unknown
`apply_prior_knowledge_softly`	Whether to apply prior knowledge softly (logical)
`reg_method`	Regression method for adjacency matrix estimation. "ols": ordinary least squares, "lasso": LASSO regression, "adaptive_lasso": adaptive LASSO regression (default), "ridge": Ridge regression (robust to multicollinearity; does not perform sparse estimation).
`lambda`	LASSO penalty (lambda) selection. Same choices as `lingam_direct()`.
`init_method`	Method for estimating the initial weights of adaptive LASSO regression ("ols" (default) or "ridge").

Details

Unlike lingam_direct(), this function has no measure argument: the multi-group causal-order search only supports the pwling (pairwise likelihood / entropy approximation) objective, matching the upstream Python MultiGroupDirectLiNGAM, which does not offer a kernel-based multi-group search.

For downstream analysis of a single group (total causal effects, independence tests of residuals, plotting), extract that group as a plain LingamResult with get_group_result() and pass it to the existing single-group functions (estimate_total_effect(), estimate_all_total_effects(), get_error_independence_p_values(), plot_adjacency(), autoplot(), tidy()); no multi-group-specific wrappers are provided for these.

Value

A MultiGroupLingamResult object (list) containing:

- adjacency_matrices: a named list of adjacency matrices, one per group (name = group name). Each follows the same convention as lingam_direct(): B[i, j] is the causal coefficient from variable j to variable i (j -> i).
- causal_order: the causal order shared by all groups (integer vector of 1-based indices).

References

S. Shimizu. Joint estimation of linear non-Gaussian acyclic models. Neurocomputing, 81: 104-107, 2012.

Examples

```
mg <- generate_multi_group_sample()
res <- lingam_multi_group(mg$data_list, reg_method = "ols")
print(res)

# Analyze group 1 with the existing single-group tooling
g1 <- get_group_result(res, 1)
estimate_all_total_effects(mg$data_list[[1]], g1, method = "ols")
```

---

lingam_multi_group_bootstrap

Bootstrap for Multi-Group Direct LiNGAM

Description

Bootstrap for Multi-Group Direct LiNGAM

Usage

```
lingam_multi_group_bootstrap(
  X_list,
  n_sampling,
  prior_knowledge = NULL,
  apply_prior_knowledge_softly = FALSE,
  reg_method = "adaptive_lasso",
  lambda = "BIC",
  init_method = "ols",
  seed = NULL,
  verbose = TRUE,
  parallel = FALSE,
  n_cores = NULL,
  compute_total_effects = TRUE
)
```

Arguments

X_list	A list of numeric matrices or data frames (length >= 2), one per group. Same requirements as lingam_multi_group().
n_sampling	Number of bootstrap iterations
prior_knowledge	Prior knowledge matrix (NULL allowed). Applied to every group, same as lingam_multi_group().
apply_prior_knowledge_softly	Apply prior knowledge softly (logical)
reg_method	Regression method ("ols", "lasso", "adaptive_lasso", "ridge")
lambda	Lambda selection ("lambda.min", "lambda.1se", "AIC", "BIC", "oracle")
init_method	Method for estimating the initial weights of adaptive LASSO regression ("ols" or "ridge")
seed	Random seed (NULL allowed)
verbose	Whether to display progress (logical)
parallel	Whether to use parallel processing (logical)
n_cores	Number of cores to use (integer, NULL allowed). When NULL, the number of cores is limited to a maximum of 2 for safety. Ignored when parallel = FALSE.
compute_total_effects	Whether to also compute total causal effects for every variable pair on each bootstrap iteration (logical, default TRUE). Set to FALSE to skip it when only edge/order stability is needed.

Details

Each element of the returned list is a regular BootstrapResult, so the existing single-group bootstrap functions (get_probabilities(), get_causal_direction_counts(), get_directed_acyclic_graph_counts(), get_total_causal_effects(), get_causal_order_stability(), plot_bootstrap_probabilities(),

tidy()) all work by extracting a group with result[[group_name]] or result[[i]], mirroring the upstream Python API (which returns a list of BootstrapResult per group).

Total effects use path products, not regression. Each bootstrap iteration's total-effect matrix is the sum of path-coefficient products over the DAG defined by that iteration's adjacency matrix (matching the upstream Python MultiGroup bootstrap), which is a different method from lingam_direct_bootstrap()'s regression-based estimate_all_total_effects().

On iteration failures: as in lingam_direct_bootstrap(), each iteration is wrapped in tryCatch(); a failing iteration is skipped with a warning, and only if every iteration fails is an error raised.

On reproducibility: same policy as lingam_direct_bootstrap(). During parallel execution, L'Ecuyer parallel random number streams via parallel::clusterSetRNGStream() are used. Results are reproducible given the same seed and same n_cores, but they do not numerically match the results of sequential execution (parallel = FALSE). If you need results that exactly match the sequential version, use parallel = FALSE.

Value

A MultiGroupBootstrapResult: a named list (one element per group) of BootstrapResult objects (see lingam_direct_bootstrap()), with class "MultiGroupBootstrapResult".

Examples

```
mg <- generate_multi_group_sample()

bs <- lingam_multi_group_bootstrap(mg$data_list,
  n_sampling = 10L,
  reg_method = "ols",
  seed = 42
)
get_probabilities(bs[[1]])


bs_par <- lingam_multi_group_bootstrap(mg$data_list,
  n_sampling = 30L,
  reg_method = "ols",
  seed = 42,
  parallel = TRUE,
  n_cores = 2L
)
```

---

lingam_parce *Bottom-Up ParceLiNGAM*

Description

A causal ordering method robust against latent confounders. Unlike lingam_direct(), which always returns a full causal order, this algorithm searches from the sink (most downstream) side and stops as soon as an independence test is rejected. Variables it could not order are returned together as

a single "unresolved block" (suspected to share a latent confounder), and the corresponding entries of the adjacency matrix are set to NA rather than estimated.

Usage

```
lingam_parce(
  X,
  alpha = 0.1,
  prior_knowledge = NULL,
  independence = "hsic",
  ind_corr = 0.5,
  reg_method = "adaptive_lasso",
  lambda = "BIC",
  init_method = "ols"
)
```

Arguments

X	Numeric matrix (n_samples x n_features), data frame or matrix
alpha	Significance level for the independence test. alpha = 0 disables rejection entirely, so a full causal order is always returned (equivalent in spirit to lingam_direct(), but using the ParceLiNGAM sink-search direction and regression). Must be non-negative.
prior_knowledge	Prior knowledge matrix (n_features x n_features) or NULL. 0: no directed path from x_i to x_j 1: directed path from x_i to x_j -1: unknown
independence	Independence measure used for the ordering search: "hsic" (default) uses the HSIC gamma-approximation test combined across explanatory variables via Fisher's method; "fcorr" uses the F-correlation (kernel canonical correlation) and rejects based on ind_corr instead of a p-value.
ind_corr	Threshold on the F-correlation value used only when independence = "fcorr": a candidate is rejected once the largest F-correlation against any explanatory variable is at or above this value. Must be non-negative. Ignored when independence = "hsic".
reg_method	Regression method for adjacency matrix estimation. "ols": ordinary least squares, "lasso": LASSO regression, "adaptive_lasso": adaptive LASSO regression (default, matches the upstream Python implementation's predict_adaptive_lasso), "ridge": Ridge regression.
lambda	LASSO penalty (lambda) selection. Same options as lingam_direct(): "lambda.min", "lambda.1se", "AIC", "BIC" (default), "oracle" (adaptive LASSO only).
init_method	Method for estimating the initial weights of adaptive LASSO regression ("ols" (default) or "ridge").

Details

Because HSIC forms full n x n Gram matrices, it is O(n^2) per test; avoid very large n (beyond a few thousand) with independence = "hsic".

independence = "fcorr" rejects based on the raw F-correlation value (ind_corr), not a p-value, so it is not directly comparable to alpha.

get_error_independence_p_values_parce() uses the HSIC test rather than the correlation-based test used by get_error_independence_p_values() for LingamResult objects.

lingam_parce_bootstrap() treats NA (unresolved) edges as absent when aggregating, and does not support get_causal_order_stability() (see its documentation for details). This function does not expose a regressor or bw_method argument, unlike the upstream Python implementation.

Value

A ParceLingamResult object (list) containing:

- adjacency_matrix: adjacency matrix B (n_features x n_features). **Convention: B[i, j] is the causal coefficient from variable j to variable i (j -> i)**, same as lingam_direct(). Entries between two variables that ended up in the same unresolved block are NA.
- causal_order: a list of integer vectors. Elements of length 1 are variables with a fully resolved position; an element of length > 1 (at most one, always first) is the unresolved block. Earlier elements are more upstream.
- p_values: independence-test p-values (or F-correlation values, for independence = "fcorr") for each step that successfully placed a variable, in the order variables were placed (diagnostic only).
- independence: the independence measure used.

References

Tashiro, T., Shimizu, S., Hyvarinen, A., and Washio, T. (2014). ParceLiNGAM: a causal ordering method robust against latent confounders. Neural Computation, 26(1), 57-83.

Examples

```
confounded <- generate_parce_sample(n = 500, seed = 1)

result <- lingam_parce(confounded$data, reg_method = "ols")
print(result)

# The variable pair sharing the latent confounder is left unresolved (NA)
result$adjacency_matrix[confounded$confounded_pair, confounded$confounded_pair]

# Total effect estimation warns and returns NA for confounded variables
estimate_total_effect_parce(confounded$data, result,
  from_index = confounded$confounded_pair[1], to_index = 1
)
```

lingam_parce_bootstrap *Bootstrap for Bottom-Up ParceLiNGAM*

Description

Bootstrap for Bottom-Up ParceLiNGAM

Usage

```
lingam_parce_bootstrap(
  X,
  n_sampling,
  prior_knowledge = NULL,
  alpha = 0.1,
  independence = "hsic",
  ind_corr = 0.5,
  reg_method = "adaptive_lasso",
  lambda = "BIC",
  init_method = "ols",
  seed = NULL,
  verbose = TRUE,
  parallel = FALSE,
  n_cores = NULL,
  compute_total_effects = TRUE
)
```

Arguments

X	Numeric matrix (n_samples x n_features)
n_sampling	Number of bootstrap iterations
prior_knowledge	Prior knowledge matrix (NULL allowed)
alpha	Significance level, passed to lingam_parce()
independence	Independence measure, passed to lingam_parce()
ind_corr	F-correlation rejection threshold, passed to lingam_parce()
reg_method	Regression method ("ols", "lasso", "adaptive_lasso", "ridge")
lambda	Lambda selection ("lambda.min", "lambda.1se", "AIC", "BIC", "oracle")
init_method	Method for estimating the initial weights of adaptive LASSO regression ("ols" or "ridge")
seed	Random seed (NULL allowed)
verbose	Whether to display progress (logical)
parallel	Whether to use parallel processing (logical)

`n_cores` Number of cores to use (integer, NULL allowed)

`compute_total_effects`
Whether to also estimate total causal effects for every variable pair on each bootstrap iteration (logical, default `TRUE`).

Details

Total effects are path sums, not regression estimates. Each iteration's total-effect matrix is built from `calculate_total_effect()` (summing products of adjacency-matrix coefficients along every directed path), matching the upstream Python implementation's bootstrap method (`estimate_total_effect2`). If a variable's row in the adjacency matrix contains `NA` (it is part of an unresolved block), all of its outgoing total effects are set to `NA` for that iteration, since its causal parents cannot be identified.

`NA` **(unresolved) edges are treated as absent when aggregating.** Both the adjacency matrix and the total-effect matrix have `NA` replaced by `0` before being stored in the returned `BootstrapResult`, matching the numpy comparison semantics used by the upstream implementation (where `np.abs(nan) > threshold` evaluates to `FALSE`). This means, for example, `get_probabilities()` reports the confounded pair's edge probability as the fraction of resamples in which the order happened to resolve, not as `NA`.

`causal_orders` **is not populated** (unlike `lingam_direct_bootstrap()`): ParceLiNGAM's causal order can include an unresolved block, which does not fit the fixed-length integer-vector format `causal_orders` requires. As a result, `get_causal_order_stability()` cannot be used with a `BootstrapResult` returned by this function.

Value

A `BootstrapResult` (list); see `lingam_direct_bootstrap()` for the query helpers that operate on it (`get_probabilities()`, `get_causal_direction_counts()`, `get_directed_acyclic_graph_counts()`, `get_total_causal_effects()`).

Examples

```
confounded <- generate_parce_sample(n = 500, seed = 1)

bs <- lingam_parce_bootstrap(confounded$data,
  n_sampling = 10L,
  reg_method = "ols",
  seed = 42
)
get_probabilities(bs)
```

`lingam_rcd` *RCD (Repetitive Causal Discovery)*

Description

A causal discovery method robust against latent confounders. Unlike `lingam_direct()` or `lingam_parce()`, RCD does not attempt to recover a full or partial causal order. Instead, it repeatedly extracts each variable's **ancestor set** by scanning variable subsets of increasing size (`extract_ancestors()`), narrows each ancestor set down to direct **parents** (`extract_parents()`), and finally tests remaining parent-free pairs for a shared latent confounder (`extract_vars_sharing_confounders()`). Pairs found to share a latent confounder are marked `NA` in the adjacency matrix rather than estimated.

Usage

```
lingam_rcd(
  X,
  max_explanatory_num = 2L,
  cor_alpha = 0.01,
  ind_alpha = 0.01,
  shapiro_alpha = 0.01,
  MLHSICR = FALSE,
  independence = "hsic",
  ind_corr = 0.5
)
```

Arguments

`X`	Numeric matrix (n_samples x n_features), data frame or matrix
`max_explanatory_num`	Maximum number of explanatory variables considered when searching for ancestors (i.e. the search scans variable subsets of size up to `max_explanatory_num + 1`). Larger values increase statistical power but grow combinatorially in cost. Must be an integer of 1 or more.
`cor_alpha`	Significance level for the Pearson correlation tests used throughout the algorithm (ancestor-subset screening, parent extraction, confounder-pair detection). Must be non-negative.
`ind_alpha`	Significance level for the HSIC independence test (used when `independence = "hsic"`). Must be non-negative.
`shapiro_alpha`	Significance level for the Shapiro-Wilk non-Gaussianity test used when screening candidate ancestor subsets. Must be non-negative.
`MLHSICR`	If `TRUE`, falls back to HSIC-sum-minimizing regression (instead of OLS) when the OLS residual is not independent of the explanatory variables and more than one explanatory variable is present. Substantially increases computation time.
`independence`	Independence measure used for the sink search: "hsic" (default) uses the HSIC gamma-approximation test; "fcorr" uses the F-correlation (kernel canonical correlation) and rejects based on `ind_corr` instead of a p-value.
`ind_corr`	Threshold on the F-correlation value, used only when `independence = "fcorr"`. Must be non-negative. Ignored when `independence = "hsic"`.

Details

The algorithm has three stages: (1) `extract_ancestors()` grows each variable's ancestor set by repeatedly scanning variable subsets; (2) `extract_parents()` narrows ancestor sets down to direct parents; (3) `extract_vars_sharing_confounders()` tests remaining parent-free pairs for a shared latent confounder. `NA` entries in `adjacency_matrix` mean the corresponding pair is suspected to share a latent confounder, not that no relationship was estimated.

`max_explanatory_num` controls both statistical power and computational cost: stage 1 scans `choose(n_features, k)` subsets for each subset size `k` up to `max_explanatory_num + 1`, and each subset requires several HSIC tests (each O(n^2) in the sample size when `independence = "hsic"`). Cost grows quickly with both the number of variables and `n`.

`MLHSICR = TRUE` replaces the OLS residual in the independence check with a residual obtained by directly minimizing the sum of HSIC statistics between the residual and each explanatory variable via numerical optimization (`stats::optim(method = "L-BFGS-B")`). This can recover independence in cases where OLS cannot, but requires re-optimizing for every candidate subset where the OLS residual fails, and is therefore substantially slower.

The Shapiro-Wilk test (`stats::shapiro.test()`) used for the non-Gaussianity check is limited to `n` between 3 and 5000. For `n` above 5000, a deterministic evenly-spaced subsample of 5000 observations is tested instead (same policy as `test_residual_normality()`), so results remain reproducible without touching the RNG state. This subsampling has no effect when `n <= 5000`.

This function does not expose a `bw_method` argument (kernel widths are always the median heuristic; see `hsic_kernel_width()`), unlike some upstream implementations. `lingam_rcd_bootstrap()` does not support `get_causal_order_stability()`, since RCD has no causal order.

Value

An `RCDResult` object (list) containing:

- `adjacency_matrix`: adjacency matrix B (n_features x n_features). **Convention: `B[i, j]` is the causal coefficient from variable j to variable i (j -> i)**, same as `lingam_direct()`. Entries between two variables found to share a latent confounder are `NA`.
- `ancestors_list`: a list of length `n_features`; element `i` is the sorted integer vector of variables found to be ancestors of variable `i` (possibly empty). Unlike `lingam_parce()`, there is no `causal_order`: RCD estimates ancestor relations directly rather than a total or partial order.

References

Maeda, T. N. and Shimizu, S. (2020). RCD: Repetitive causal discovery of linear non-Gaussian acyclic models with latent confounders. AISTATS 2020, PMLR 108: 735-745.

Examples

```
confounded <- generate_rcd_sample(n = 300, seed = 1)

result <- lingam_rcd(confounded$data)
print(result)

# The variable pair sharing the latent confounder is left NA
```

```
result$adjacency_matrix[confounded$confounded_pair, confounded$confounded_pair]

# Total effect estimation warns and returns NA for confounded variables
estimate_total_effect_rcd(confounded$data, result,
  from_index = confounded$confounded_pair[1], to_index = 1
)
```

`lingam_rcd_bootstrap` *Bootstrap for RCD*

Description

Bootstrap for RCD

Usage

```
lingam_rcd_bootstrap(
  X,
  n_sampling,
  max_explanatory_num = 2L,
  cor_alpha = 0.01,
  ind_alpha = 0.01,
  shapiro_alpha = 0.01,
  MLHSICR = FALSE,
  independence = "hsic",
  ind_corr = 0.5,
  seed = NULL,
  verbose = TRUE,
  parallel = FALSE,
  n_cores = NULL,
  compute_total_effects = TRUE
)
```

Arguments

`X`	Numeric matrix (n_samples x n_features)
`n_sampling`	Number of bootstrap iterations
`max_explanatory_num`	Maximum number of explanatory variables, passed to `lingam_rcd()`
`cor_alpha`	Significance level for correlation tests, passed to `lingam_rcd()`
`ind_alpha`	Significance level for the HSIC independence test, passed to `lingam_rcd()`
`shapiro_alpha`	Significance level for the non-Gaussianity test, passed to `lingam_rcd()`
`MLHSICR`	Whether to use MLHSICR regression, passed to `lingam_rcd()`
`independence`	Independence measure, passed to `lingam_rcd()`
`ind_corr`	F-correlation rejection threshold, passed to `lingam_rcd()`

seed	Random seed (NULL allowed)
verbose	Whether to display progress (logical)
parallel	Whether to use parallel processing (logical)
n_cores	Number of cores to use (integer, NULL allowed)
compute_total_effects	Whether to also estimate total causal effects for every ancestor pair on each bootstrap iteration (logical, default TRUE).

Details

Total effects are computed only for ancestor pairs, driven by each iteration's ancestors_list (unlike lingam_direct_bootstrap() and lingam_parce_bootstrap(), which loop over all variable pairs). For a pair (from, to) with from in to's ancestor set, the total effect is obtained via calculate_total_effect() (summing products of adjacency-matrix coefficients along every directed path). If from's row in the adjacency matrix contains NA (it shares a latent confounder with some other variable), the effect is set to NA for that iteration.

NA **(confounded) edges are treated as absent when aggregating.** Both the adjacency matrix and the total-effect matrix have NA replaced by 0 before being stored in the returned BootstrapResult, matching the policy used by lingam_parce_bootstrap().

causal_orders **is not populated**: RCD has no causal order (see lingam_rcd()). As a result, get_causal_order_stability() cannot be used with a BootstrapResult returned by this function.

RCD's fit step is HSIC-heavy and can be slow per iteration, especially with MLHSICR = TRUE; keep n_sampling modest in examples.

Value

A BootstrapResult (list); see lingam_direct_bootstrap() for the query helpers that operate on it (get_probabilities(), get_causal_direction_counts(), get_directed_acyclic_graph_counts(), get_total_causal_effects()).

Examples

```
confounded <- generate_rcd_sample(n = 300, seed = 1)

bs <- lingam_rcd_bootstrap(confounded$data,
  n_sampling = 5L,
  seed = 42
)
get_probabilities(bs)
```

lingam_var *VAR-LiNGAM for time series causal discovery*

Description

Fits a vector autoregressive (VAR) model to time series data and applies Direct LiNGAM to the residuals to recover the instantaneous (lag-0) causal structure. The lagged causal matrices are then derived from the VAR coefficients and the instantaneous structure.

Usage

```
lingam_var(
  X,
  lags = 1L,
  criterion = "bic",
  measure = "pwling",
  reg_method = "adaptive_lasso",
  lambda = "BIC",
  init_method = "ols",
  prune = TRUE
)
```

Arguments

X	numeric matrix or data frame (n_samples x n_features). Rows are ordered in time (earliest first).
lags	maximum lag order. When criterion is not NULL, the best lag in 1:lags is selected by the information criterion; otherwise lags is used directly.
criterion	lag-selection criterion ("bic", "aic", "hqic", or "fpe"), or NULL to use lags directly without selection.
measure	independence measure passed to lingam_direct() ("pwling" or "kernel").
reg_method	regression method for the instantaneous adjacency matrix: "adaptive_lasso" (default), "lasso", "ols", or "ridge" (see lingam_direct()).
lambda	penalty (lambda) selection for the instantaneous matrix: "BIC" (default), "AIC", "lambda.min", "lambda.1se", or "oracle" (see lingam_direct()).
init_method	initial-weight method for adaptive LASSO (see lingam_direct()).
prune	logical; if TRUE (default, matching the Python reference), all adjacency matrices (instantaneous B0 and the lagged B_k) are refined together by adaptive LASSO so weak edges are shrunk toward zero. Requires the glmnet package. Set FALSE to keep the raw B_k = (I - B0) M_k matrices (no glmnet needed when reg_method = "ols").

Details

The model is `X_t = B0 X_t + sum_{k=1}^{p} B_k X_{t-k} + e_t`, where B0 is the instantaneous effect matrix (strictly acyclic) and e_t are mutually independent non-Gaussian errors. VAR coefficients `M_k` are estimated by ordinary least squares (no intercept); residuals `e_t = X_t - sum M_k X_{t-k}` are passed to `lingam_direct()` to obtain B0, and the lagged matrices follow `B_k = (I - B0) M_k`.

Value

A `VARLiNGAMResult` object (list) containing:

- `adjacency_matrices`: array (1 + lags, n_features, n_features). The first slice `[1, , ]` is the instantaneous matrix B0; slice `[k + 1, , ]` is the lagged matrix B_k for lag k (k = 1..lags). Convention: `B[i, j]` is the effect from variable j to variable i.
- `causal_order`: estimated causal order of the instantaneous structure (1-based indices).
- `residuals`: VAR residuals (n_samples - lags, n_features).
- `lags`: the lag order actually used.

References

Hyvärinen, A., Zhang, K., Shimizu, S., & Hoyer, P. O. (2010). Estimation of a structural vector autoregression model using non-Gaussianity. *Journal of Machine Learning Research*, 11, 1709-1731. Ported from the Python implementation cdt15/lingam (https://github.com/cdt15/lingam). See also the VARLiNGAM R code of Moneta et al. (https://sites.google.com/site/dorisentner/publications/VARLiNGAM).

Examples

```
sample <- generate_varlingam_sample(n = 500, seed = 42)

# OLS instantaneous structure without pruning (no extra packages required)
model <- lingam_var(sample$data, lags = 1, reg_method = "ols", prune = FALSE)
round(model$adjacency_matrices[1, , ], 2)  # instantaneous B0
```

---

lingam_var_bootstrap *Bootstrap for VAR-LiNGAM*

Description

Evaluates the statistical reliability of the estimated time-series DAG by resampling. Unlike the i.i.d. row resampling used for Direct LiNGAM, this uses a **residual bootstrap**: the VAR is fitted once on the original data, the residuals are resampled with replacement, and a new series is rebuilt by the VAR recursion before re-estimating VAR-LiNGAM on it (this preserves the autoregressive structure). Port of the Python reference `VARLiNGAM.bootstrap`.

Usage

```
lingam_var_bootstrap(
  X,
  n_sampling,
  lags = 1L,
  criterion = "bic",
  measure = "pwling",
  reg_method = "adaptive_lasso",
  lambda = "BIC",
  init_method = "ols",
  prune = TRUE,
  seed = NULL,
  verbose = TRUE,
  parallel = FALSE,
  n_cores = NULL
)
```

Arguments

X	numeric matrix or data frame (n_samples x n_features), rows ordered in time.
n_sampling	number of bootstrap iterations (positive integer).
lags	maximum lag order. When criterion is not NULL, the lag is selected once on the original data and then fixed across all iterations.
criterion	lag-selection criterion ("bic", "aic", "hqic", "fpe") or NULL to use lags directly.
measure	independence measure for lingam_direct() ("pwling"/"kernel").
reg_method	regression method for the instantaneous matrix.
lambda	penalty selection (see lingam_direct()).
init_method	initial-weight method for adaptive LASSO.
prune	logical; passed to lingam_var() on each iteration (default TRUE).
seed	random seed (NULL allowed).
verbose	whether to print progress (logical).
parallel	whether to distribute iterations across cores (logical).
n_cores	number of cores (integer or NULL; NULL caps at 2 for safety).

Details

Reproducibility follows the same rules as lingam_direct_bootstrap(): with parallel = TRUE, L'Ecuyer streams via parallel::clusterSetRNGStream() make results reproducible for a given seed and n_cores, but they do not match the sequential (parallel = FALSE) results.

On iteration failures: as in lingam_direct_bootstrap(), each iteration runs inside a tryCatch(); a failing iteration is reported as a warning and excluded from the result instead of aborting the run. An error is raised only if every iteration fails.

Value

a VARBootstrapResult object.

Examples

```
s <- generate_varlingam_sample(n = 500, seed = 42)

# Fast example: OLS instantaneous structure, no pruning (no glmnet needed)
bs <- lingam_var_bootstrap(s$data,
  n_sampling = 10L, lags = 1, criterion = NULL,
  reg_method = "ols", prune = FALSE, seed = 1, verbose = FALSE
)
get_var_probabilities(bs)
```

`make_prior_knowledge` *Create a prior knowledge matrix*

Description

Create a prior knowledge matrix

Usage

```
make_prior_knowledge(
  n_variables,
  exogenous_variables = NULL,
  sink_variables = NULL,
  paths = NULL,
  no_paths = NULL,
  labels = NULL
)
```

Arguments

`n_variables`	Number of variables
`exogenous_variables`	Exogenous variables (1-based index or variable name, NULL allowed) The specified variables are assumed not to be influenced by any other variable
`sink_variables`	Sink variables (1-based index or variable name, NULL allowed) The specified variables are assumed not to influence any other variable
`paths`	Variable pairs that have a directed path (NULL allowed) Of the form list(c(from, to), ...). Specified by index or variable name
`no_paths`	Variable pairs that have no directed path (NULL allowed) Of the form list(c(from, to), ...). Specified by index or variable name
`labels`	Vector of variable names (NULL allowed) Required when specifying by variable name. Pass e.g. colnames() of a data.frame

Value

Prior knowledge matrix (n_variables x n_variables) -1: unknown, 0: no path, 1: path exists

Examples

```
# Specify by index
pk <- make_prior_knowledge(6, exogenous_variables = c(4))

# Specify by variable name
pk <- make_prior_knowledge(6,
  exogenous_variables = "x3",
  sink_variables = c("x1", "x4"),
  paths = list(c("x3", "x0"), c("x3", "x2")),
  no_paths = list(c("x5", "x2")),
  labels = c("x0", "x1", "x2", "x3", "x4", "x5")
)
```

---

plot_adjacency *Plot a causal graph from an adjacency matrix with DiagrammeR*

Description

Plot a causal graph from an adjacency matrix with DiagrammeR

Usage

```
plot_adjacency(
  B,
  labels = NULL,
  threshold = 0,
  rankdir = "TB",
  title = "Estimated Causal Structure",
  shape = "circle",
  fillcolor = "lightyellow",
  bordercolor = "black",
  fontsize_node = 14,
  fontsize_edge = 10,
  edge_color = "gray40",
  edge_label_color = "red",
  true_B = NULL,
  color_tp = "forestgreen",
  color_fp = "firebrick",
  color_fn = "darkorange",
  debug = FALSE
)
```

Arguments

B	Adjacency matrix (n_features x n_features). **Convention: `B[i, j]` is the causal coefficient from variable j to variable i (j -> i).** The `adjacency_matrix` from `lingam_direct()` can be passed directly.

labels	Vector of variable names (if NULL, x0, x1, ... are generated automatically)
threshold	Minimum absolute coefficient value to display (default: 0)
rankdir	Layout direction (default: "LR") "LR" = left -> right, "RL" = right -> left, "TB" = top -> bottom, "BT" = bottom -> top
title	Graph title (default: "Estimated Causal Structure")
shape	Node shape (default: "circle") "circle", "box", "ellipse", "diamond", "plaintext", "square", "triangle", "hexagon", "octagon", etc.
fillcolor	Node fill color (default: "lightyellow")
bordercolor	Border color
fontsize_node	Node font size (default: 14)
fontsize_edge	Edge label font size (default: 10)
edge_color	Edge color (default: "gray40"). Unused when true_B is specified.
edge_label_color	Edge label color (default: "red"). Unused when true_B is specified.
true_B	True adjacency matrix (may be NULL). When specified, edges are classified into three colors: • Correct edges (estimated and true): solid line in color_tp • False positives (estimated but not true): solid line in color_fp • Missed edges (not estimated but true): dashed line in color_fn (showing the true coefficient)
color_tp	Color for correct edges (default: "forestgreen")
color_fp	Color for false-positive edges (default: "firebrick")
color_fn	Color for missed edges (default: "darkorange")
debug	Enable debug mode (logical)

Value

A grViz object (when DiagrammeR is available)

Examples

```
if (requireNamespace("DiagrammeR", quietly = TRUE)) {
  LiNGAM_sample_1000 <- generate_lingam_sample_6()

  LiNGAM_sample_1000$true_adjacency |>
    plot_adjacency(title = "True Causal Structure")

  model <- LiNGAM_sample_1000$data |>
    lingam_direct(reg_method = "ols")

  model$adjacency_matrix |>
    plot_adjacency()


  # Compare with the true structure
```

```
  # (correct = green, false positive = red, missed = orange dashed)
  model$adjacency_matrix |>
    plot_adjacency(true_B = LiNGAM_sample_1000$true_adjacency)

}
```

`plot_bootstrap_probabilities` *Draw bootstrap probabilities with DiagrammeR*

Description

Draw bootstrap probabilities with DiagrammeR

Usage

```
plot_bootstrap_probabilities(
  result,
  labels = NULL,
  min_causal_effect = NULL,
  min_probability = 0.5,
  rankdir = "TB",
  shape = "circle"
)
```

Arguments

`result`	BootstrapResult object
`labels`	Vector of variable names (NULL allowed)
`min_causal_effect`	Minimum causal effect to display
`min_probability`	Minimum probability to display
`rankdir`	Layout direction
`shape`	Node shape

Value

grViz object

Examples

```
if (requireNamespace("DiagrammeR", quietly = TRUE)) {
  LiNGAM_sample_1000 <- generate_lingam_sample_6()

  bs_model <- lingam_direct_bootstrap(LiNGAM_sample_1000$data,
    n_sampling = 30L, reg_method = "ols", seed = 42
```

```
  )
  plot_bootstrap_probabilities(bs_model)
}
```

`plot_residual_qq` *plot QQ*

Description

plot QQ

Usage

```
plot_residual_qq(X, lingam_result, ncol = 3, nrow = NULL)
```

Arguments

`X`	original data (matrix or data.frame)
`lingam_result`	return value of lingam_direct()
`ncol`	Number of columns.
`nrow`	Number of rows.

Value

A ggplot2::ggplot object with QQ plots of residuals.

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  # Load the sample data
  LiNGAM_sample_1000 <- generate_lingam_sample_6()

  # Run Direct LiNGAM
  result <- lingam_direct(LiNGAM_sample_1000$data, reg_method = "ols")

  plot_residual_qq(LiNGAM_sample_1000$data, result)
}
```

plot_varlingam_residual_qq *Q-Q plots of VAR-LiNGAM residuals*

Description

Draws per-variable normal Q-Q plots of the residuals (analogous to the Moneta `Gauss_Stats` visual check). Deviations from the reference line indicate non-Gaussianity, which supports the LiNGAM assumption. Requires ggplot2.

Usage

```
plot_varlingam_residual_qq(
  result,
  on = c("innovations", "var"),
  ncol = 3,
  nrow = NULL
)
```

Arguments

`result`	a `VARLiNGAMResult` from `lingam_var()`
`on`	which series to plot: "innovations" (default) or "var"
`ncol`	number of facet columns
`nrow`	number of facet rows (NULL = automatic)

Value

a ggplot object

References

Analogous to the residual visual check (Gauss_Stats) in the VARLiNGAM R code of Moneta, A., Entner, D., Hoyer, P. O., & Coad, A. (2013), *Oxford Bulletin of Economics and Statistics*, 75(5), 705-730. https://sites.google.com/site/dorisentner/publications/VARLiNGAM

Examples

```
s <- generate_varlingam_sample(n = 1000, seed = 42)
m <- lingam_var(s$data, lags = 1, reg_method = "ols", prune = FALSE)

plot_varlingam_residual_qq(m)
```

print.BootstrapResult *Display the contents of a BootstrapResult*

Description

Display the contents of a BootstrapResult

Usage

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

Arguments

x	BootstrapResult object
...	Additional arguments (for S3 method compatibility)

Value

The input object x, invisibly.

Examples

```
LiNGAM_sample_1000 <- generate_lingam_sample_6()

bs_model <- lingam_direct_bootstrap(LiNGAM_sample_1000$data,
  n_sampling = 30L, reg_method = "ols", seed = 42
)

print(bs_model)
```

print.causal_order_stability *print method for causal_order_stability*

Description

print method for causal_order_stability

Usage

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

Arguments

x	A causal_order_stability object
...	Additional arguments (unused)

Value

The input object x, invisibly.

Examples

```
dat <- generate_lingam_sample_6()
bs <- lingam_direct_bootstrap(dat$data, n_sampling = 30L, reg_method = "ols", seed = 42)
print(get_causal_order_stability(bs, labels = names(dat$data)))
```

---

print.ImputationBootstrapResult

Print method for ImputationBootstrapResult

Description

Print method for ImputationBootstrapResult

Usage

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

Arguments

x	ImputationBootstrapResult object
...	Additional arguments (unused)

Value

The input object x, invisibly.

Examples

```
set.seed(1)
sample6 <- generate_lingam_sample_6(n = 300, seed = 1)
X <- sample6$data
X$x5[sample.int(nrow(X), size = 30)] <- NA

if (requireNamespace("mice", quietly = TRUE)) {
  res <- bootstrap_with_imputation(X,
    n_sampling = 5L, n_repeats = 3L, seed = 42, verbose = FALSE
  )
  print(res)
}
```

print.LiMResult *Print method for LiMResult*

Description

Print method for LiMResult

Usage

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

Arguments

x	LiMResult object
digits	Number of digits to display
...	Additional arguments (unused)

Value

The input object x, invisibly.

Examples

```
set.seed(1)
dat <- generate_lim_sample(n = 300)
result <- lingam_lim(dat$data, is_continuous = dat$is_continuous)
print(result)
```

print.LingamResult *Print method for LingamResult*

Description

Print method for LingamResult

Usage

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

Arguments

x	LingamResult object
digits	Number of digits to display
...	Additional arguments (unused)

Value

The input object x, invisibly.

Examples

```
LiNGAM_sample_1000 <- generate_lingam_sample_6()
result <- lingam_direct(LiNGAM_sample_1000$data, reg_method = "ols")
print(result)
```

---

print.lingam_normality_test

Print method for lingam_normality_test

Description

Print method for lingam_normality_test

Usage

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

Arguments

x	lingam_normality_test object
...	additional arguments

Value

The input object x, invisibly.

Examples

```
LiNGAM_sample_1000 <- generate_lingam_sample_6()
result <- lingam_direct(LiNGAM_sample_1000$data, reg_method = "ols")
print(test_residual_normality(LiNGAM_sample_1000$data, result))
```

print.lingam_summary *print method for lingam_summary*

Description

print method for lingam_summary

Usage

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

Arguments

x	A lingam_summary object
...	Additional arguments (unused)

Value

The input object x, invisibly.

Examples

```
LiNGAM_sample_1000 <- generate_lingam_sample_6()
model <- lingam_direct(LiNGAM_sample_1000$data, reg_method = "ols")
print(summary_lingam(LiNGAM_sample_1000$data, model))
```

print.MultiGroupBootstrapResult *Print method for MultiGroupBootstrapResult*

Description

Print method for MultiGroupBootstrapResult

Usage

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

Arguments

x	MultiGroupBootstrapResult object
...	Additional arguments (unused)

Value

The input object x, invisibly.

Examples

```
mg <- generate_multi_group_sample()
bs <- lingam_multi_group_bootstrap(mg$data_list,
  n_sampling = 10L, reg_method = "ols", seed = 42
)
print(bs)
```

---

print.MultiGroupLingamResult

Print method for MultiGroupLingamResult

Description

Print method for MultiGroupLingamResult

Usage

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

Arguments

x	MultiGroupLingamResult object
digits	Number of digits to display
...	Additional arguments (unused)

Value

The input object x, invisibly.

Examples

```
mg <- generate_multi_group_sample()
res <- lingam_multi_group(mg$data_list, reg_method = "ols")
print(res)
```

print.ParceLingamResult *Print method for ParceLingamResult*

Description

Print method for ParceLingamResult

Usage

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

Arguments

x	ParceLingamResult object
digits	Number of digits to display
...	Additional arguments (unused)

Value

The input object x, invisibly.

Examples

```
confounded <- generate_parce_sample(n = 300, seed = 42)
result <- lingam_parce(confounded$data, reg_method = "ols")
print(result)
```

print.RCDResult *Print method for RCDResult*

Description

Print method for RCDResult

Usage

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

Arguments

x	RCDResult object
digits	Number of digits to display
...	Additional arguments (unused)

Value

The input object x, invisibly.

Examples

```
confounded <- generate_rcd_sample(n = 300, seed = 1)
result <- lingam_rcd(confounded$data)
print(result)
```

`print.VARBootstrapResult`

Print a VARBootstrapResult

Description

Print a VARBootstrapResult

Usage

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

Arguments

x	a VARBootstrapResult object
...	additional arguments (unused)

Value

The input object x, invisibly.

Examples

```
s <- generate_varlingam_sample(n = 500, seed = 42)
bs <- lingam_var_bootstrap(s$data,
  n_sampling = 10L, lags = 1, criterion = NULL,
  reg_method = "ols", prune = FALSE, seed = 1, verbose = FALSE
)
print(bs)
```

print.VARLiNGAMResult *Print method for VARLiNGAMResult*

Description

Print method for VARLiNGAMResult

Usage

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

Arguments

x	VARLiNGAMResult object
digits	number of digits to display
...	additional arguments (unused)

Value

The input object x, invisibly.

Examples

```
sample <- generate_varlingam_sample(n = 500, seed = 42)
model <- lingam_var(sample$data, lags = 1, reg_method = "ols", prune = FALSE)
print(model)
```

print.var_stationarity *Print method for var_stationarity*

Description

Print method for var_stationarity

Usage

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

Arguments

x	a var_stationarity object
...	additional arguments (unused)

Value

The input object x, invisibly.

Examples

```
s <- generate_varlingam_sample(n = 1000, seed = 42)
m <- lingam_var(s$data, lags = 1, reg_method = "ols", prune = FALSE)
print(check_var_stationarity(m))
```

---

summary_lingam *Summarize the goodness-of-fit of a Direct LiNGAM model at once*

Description

For a fitted Direct LiNGAM model, this verifies how well the two main assumptions on which LiNGAM relies (mutual independence of residuals and non-Gaussianity of residuals) hold, all at once, and displays the results together. Internally it calls `get_error_independence_p_values()` and `test_residual_normality()`.

Usage

```
summary_lingam(
  X,
  lingam_result,
  independence_method = "spearman",
  normality_method = "shapiro",
  alpha = 0.05
)
```

Arguments

X	The original data (matrix or data.frame), the one used to estimate lingam_result.
lingam_result	The return value of `lingam_direct()` (a LingamResult object)
independence_method	The type of correlation coefficient used in the residual independence test ("spearman", "pearson", "kendall"). Passed to `get_error_independence_p_values()`.
normality_method	The method for the residual normality test ("shapiro", "ks", "ad", "lillie", "jb"). Passed to `test_residual_normality()`.
alpha	Significance level (default: 0.05)

Details

Gaussian-likelihood-based criteria such as BIC/AIC are not included because they are theoretically inconsistent with LiNGAM's assumption that "the errors are non-Gaussian". Instead, the verification results of the assumptions themselves are summarized.

Value

A list of class `lingam_summary`, containing the following elements:

- `n_variables`, `n_samples`: Number of variables / number of observations
- `causal_order`: Causal order (variable-name labels)
- `n_edges`: Number of nonzero elements in the adjacency matrix (number of estimated edges)
- `independence_p_values`: Matrix of p-values from the independence test between residuals
- `n_dependent_pairs`, `n_pairs`: Number of pairs with p < alpha / total number of pairs
- `min_independence_p`: Minimum p-value of the independence test
- `normality`: Result of the normality test (a `lingam_normality_test` object)
- `n_non_gaussian`: Number of variables judged to be non-Gaussian
- `alpha`, `independence_method`, `normality_method`: The settings used

Examples

```
LiNGAM_sample_1000 <- generate_lingam_sample_6()

model <- lingam_direct(LiNGAM_sample_1000$data, reg_method = "ols")

summary_lingam(LiNGAM_sample_1000$data, model)
```

---

test_residual_normality

Test normality of residuals from Direct LiNGAM

Description

Calculate residuals (error terms) from the estimated adjacency matrix and test their normality. Since LiNGAM assumes non-Gaussian errors, rejecting normality (small p-value) supports the LiNGAM model assumption.

Usage

```
test_residual_normality(X, lingam_result, method = "shapiro", alpha = 0.05)
```

Arguments

`X`	original data matrix or data.frame
`lingam_result`	result from lingam_direct()
`method`	normality test method "shapiro" : Shapiro-Wilk test (default, n <= 5000) "ks" : Kolmogorov-Smirnov test (n > 5000) "ad" : Anderson-Darling test (requires nortest package) "lillie" : Lilliefors test (requires nortest package) "jb" : Jarque-Bera test (requires tseries package)
`alpha`	significance level (default: 0.05)

Value

data.frame with test results for each variable

Examples

```
# Load the sample data
LiNGAM_sample_1000 <- generate_lingam_sample_6()

# Run Direct LiNGAM
result <- lingam_direct(LiNGAM_sample_1000$data, reg_method = "ols")

# Shapiro-Wilk (default)
test_residual_normality(LiNGAM_sample_1000$data, result)
```

---

test_varlingam_residual_normality

Test the non-Gaussianity of VAR-LiNGAM residuals

Description

LiNGAM assumes the error terms are non-Gaussian, so rejecting normality (small p-value) supports the model assumption. By default the test is run on the LiNGAM innovations `e_t = (I - B0) n_t` (the independent errors the model assumes), where `n_t` are the stored VAR residuals; set `on = "var"` to test the reduced-form VAR residuals `n_t` directly instead.

Usage

```
test_varlingam_residual_normality(
  result,
  method = "shapiro",
  alpha = 0.05,
  on = c("innovations", "var")
)
```

Arguments

`result`	a `VARLiNGAMResult` from `lingam_var()`
`method`	normality test ("shapiro", "ks", "ad", "lillie", "jb"); see `test_residual_normality()` for package requirements
`alpha`	significance level (default 0.05)
`on`	which series to test: "innovations" (default, `e_t = (I - B0) n_t`) or "var" (the reduced-form VAR residuals `n_t`)

Value

a `lingam_normality_test` data frame (one row per variable), printed via `print.lingam_normality_test()`.

References

Residual non-Gaussianity diagnostics inspired by the VARLiNGAM R code (Gauss_Tests) of Moneta, A., Entner, D., Hoyer, P. O., & Coad, A. (2013), *Oxford Bulletin of Economics and Statistics*, 75(5), 705-730. https://sites.google.com/site/dorisentner/publications/VARLiNGAM

Examples

```
s <- generate_varlingam_sample(n = 1000, seed = 42)
m <- lingam_var(s$data, lags = 1, reg_method = "ols", prune = FALSE)
test_varlingam_residual_normality(m)
```

`test_varlingam_residual_normality_all`

Run several normality tests on VAR-LiNGAM residuals at once

Description

Convenience wrapper (analogous to the Moneta `Gauss_Tests`) that applies multiple normality tests to the residuals and returns a single table with one p-value column per method plus per-variable skewness and excess kurtosis. Methods whose optional package is unavailable are skipped with a warning.

Usage

```
test_varlingam_residual_normality_all(
  result,
  methods = c("shapiro", "ad", "lillie", "jb"),
  alpha = 0.05,
  on = c("innovations", "var")
)
```

Arguments

`result`	a `VARLiNGAMResult` from `lingam_var()`
`methods`	character vector of tests to run; any of "shapiro", "ks", "ad", "lillie", "jb" (default runs shapiro/ad/lillie/jb)
`alpha`	significance level (default 0.05)
`on`	which series to test: "innovations" (default) or "var"

Value

a data frame with columns `variable`, `skewness`, `kurtosis`, one `p_<method>` column per method, and `all_non_gauss` (TRUE when every run test rejects normality for that variable).

References

Analogous to the multi-test residual check (Gauss_Tests) in the VARLiNGAM R code of Moneta, A., Entner, D., Hoyer, P. O., & Coad, A. (2013), *Oxford Bulletin of Economics and Statistics*, 75(5), 705-730. https://sites.google.com/site/dorisentner/publications/VARLiNGAM

Examples

```
s <- generate_varlingam_sample(n = 1000, seed = 42)
m <- lingam_var(s$data, lags = 1, reg_method = "ols", prune = FALSE)
test_varlingam_residual_normality_all(m, methods = c("shapiro", "jb"))
```

---

tidy.BootstrapResult *Convert a BootstrapResult to a tidy data.frame*

Description

Returns a summary of the occurrence count, proportion, and effect size for each causal direction. Internally it calls `get_causal_direction_counts()`, so that function's arguments can be passed through `...`.

Usage

```
## S3 method for class 'BootstrapResult'
tidy(x, ...)
```

Arguments

`x`	The return value of `lingam_direct_bootstrap()` (a `BootstrapResult` object)
`...`	Arguments passed to `get_causal_direction_counts()` (such as `n_directions`, `min_causal_effect`, `split_by_causal_effect_sign`, `labels`)

Value

data.frame (from, to, count, proportion, ...)

Examples

```
dat <- generate_lingam_sample_6()
bs <- lingam_direct_bootstrap(dat$data, n_sampling = 30L, reg_method = "ols", seed = 42)
tidy(bs)
```

tidy.ImputationBootstrapResult *Convert an ImputationBootstrapResult to a tidy data.frame*

Description

Collapses the imputation dimension with `as_bootstrap_result()` and then summarizes the causal direction counts like `tidy.BootstrapResult()`.

Usage

```
## S3 method for class 'ImputationBootstrapResult'
tidy(x, aggregate = c("median", "mean"), ...)
```

Arguments

`x`	The return value of `bootstrap_with_imputation()` (an `ImputationBootstrapResult` object)
`aggregate`	How to collapse the `n_repeats` imputation dimension, passed to `as_bootstrap_result()` ("median" or "mean")
`...`	Arguments passed to `get_causal_direction_counts()`

Value

data.frame (from, to, count, proportion, ...)

Examples

```
dat <- generate_lingam_sample_6(n = 200, seed = 1)$data
dat[sample(nrow(dat), 20), 1] <- NA
bs <- bootstrap_with_imputation(dat, n_sampling = 5L, n_repeats = 2L, seed = 42)
tidy(bs)
```

tidy.LiMResult *Convert a LiMResult to a tidy data.frame*

Description

Converts the estimated adjacency matrix of a LiM model into a long-format data.frame with one edge per row, exactly like `tidy.LingamResult()`.

Usage

```
## S3 method for class 'LiMResult'
tidy(x, threshold = 0, ...)
```

Arguments

x	The return value of lingam_lim() (a LiMResult object)
threshold	Coefficients with an absolute value at or below this are not treated as edges (default: 0)
...	Unused

Value

data.frame(from, to, estimate)

Examples

```
set.seed(1)
dat <- generate_lim_sample(n = 300)
model <- lingam_lim(dat$data, is_continuous = dat$is_continuous)
tidy(model)
```

---

tidy.LingamResult *Convert a LingamResult to a tidy data.frame*

Description

Converts the estimated adjacency matrix into a long-format data.frame with one edge per row. Following the B[i, j] convention (the coefficient for j -> i), the from column is the cause and the to column is the effect. Convenient for visualization with ggplot2 or ggraph and for filtering with dplyr.

Usage

```
## S3 method for class 'LingamResult'
tidy(x, threshold = 0, ...)
```

Arguments

x	The return value of lingam_direct() (a LingamResult object)
threshold	Coefficients with an absolute value at or below this are not treated as edges (default: 0)
...	Unused

Value

data.frame(from, to, estimate). from/to are variable names (strings) and estimate is the causal coefficient. Returns a 0-row data.frame if there are no edges.

Examples

```
dat <- generate_lingam_sample_6()
model <- lingam_direct(dat$data, reg_method = "ols")
tidy(model)
```

`tidy.MultiGroupBootstrapResult` *Convert a MultiGroupBootstrapResult to a tidy data.frame*

Description

Stacks each group's causal direction counts (via `get_causal_direction_counts()`) into a single data.frame with a `group` column in front. Arguments for `get_causal_direction_counts()` can be passed through `...`.

Usage

```
## S3 method for class 'MultiGroupBootstrapResult'
tidy(x, ...)
```

Arguments

`x`	The return value of `lingam_multi_group_bootstrap()` (a `MultiGroupBootstrapResult` object)
`...`	Arguments passed to `get_causal_direction_counts()` (such as `n_directions`, `min_causal_effect`, `split_by_causal_effect_sign`)

Value

data.frame (group, from, to, count, proportion, ...)

Examples

```
mg <- generate_multi_group_sample()
bs <- lingam_multi_group_bootstrap(mg$data_list,
  n_sampling = 10L, reg_method = "ols", seed = 42
)
tidy(bs)
```

tidy.MultiGroupLingamResult

Convert a MultiGroupLingamResult to a tidy data.frame

Description

Stacks the per-group edge lists into a single long-format data.frame with a `group` column in front, one edge per row per group (the causal order is shared across groups but the coefficients differ).

Usage

```
## S3 method for class 'MultiGroupLingamResult'
tidy(x, threshold = 0, ...)
```

Arguments

`x`	The return value of `lingam_multi_group()` (a `MultiGroupLingamResult` object)
`threshold`	Coefficients with an absolute value at or below this are not treated as edges (default: 0)
`...`	Unused

Value

data.frame(group, from, to, estimate)

Examples

```
mg <- generate_multi_group_sample()
model <- lingam_multi_group(mg$data_list, reg_method = "ols")
tidy(model)
```

tidy.ParceLingamResult

Convert a ParceLingamResult to a tidy data.frame

Description

Converts the estimated adjacency matrix into a long-format data.frame with one edge per row, like `tidy.LingamResult()`. `NA` entries of the adjacency matrix (variable pairs whose order could not be resolved / suspected latent confounding) are kept as rows with `estimate = NA` so they remain visible; drop them with e.g. `subset(tidy(x), !is.na(estimate))` if not needed.

Usage

```
## S3 method for class 'ParceLingamResult'
tidy(x, threshold = 0, ...)
```

Arguments

x	The return value of lingam_parce() (a ParceLingamResult object)
threshold	Coefficients with an absolute value at or below this are not treated as edges (default: 0). NA entries are always kept.
...	Unused

Value

data.frame(from, to, estimate)

Examples

```
dat <- generate_parce_sample(n = 500, seed = 42)
model <- lingam_parce(dat$data)
tidy(model)
```

---

tidy.RCDResult *Convert an RCDResult to a tidy data.frame*

Description

Converts the estimated adjacency matrix into a long-format data.frame with one edge per row, like tidy.LingamResult(). NA entries of the adjacency matrix (variable pairs suspected to share a latent confounder; marked in both directions) are kept as rows with estimate = NA so they remain visible; drop them with e.g. subset(tidy(x), !is.na(estimate)) if not needed.

Usage

```
## S3 method for class 'RCDResult'
tidy(x, threshold = 0, ...)
```

Arguments

x	The return value of lingam_rcd() (an RCDResult object)
threshold	Coefficients with an absolute value at or below this are not treated as edges (default: 0). NA entries are always kept.
...	Unused

Value

data.frame(from, to, estimate)

Examples

```
confounded <- generate_rcd_sample(n = 300, seed = 1)
model <- lingam_rcd(confounded$data)
tidy(model)
```

Index