

Package: lagdynamics (via r-universe)

July 21, 2026

Title Lag Sequential Analysis, Dynamics, and Lag Transition Networks

Version 0.32

Description A modern, tidy toolkit for lag sequential analysis and lag transition networks of categorical event and sequence data. It provides an accessible, unified workflow for fitting, inspecting, visualising, and comparing lagged transition patterns, with tidy outputs throughout. Includes confirmatory tools for uncertainty, robustness, and group differences, including bootstrap intervals, analytic certainty, split-half reliability, case-drop stability, permutation tests, and Bayesian group comparisons. Supports long-format event-log import, import from common sequence and state-sequence objects, multi-lag analysis, structural-zero constraints, transition and initial probabilities, plotting of transition structures, and a directed transfer-entropy measure. The lag sequential analysis framework follows Sackett and others (1979) [doi:10.3758/BF03205679](https://doi.org/10.3758/BF03205679).

License MIT + file LICENSE

URL <https://github.com/mohsaqr/lagdynamics>

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

Language en-US

Encoding UTF-8

RoxygenNote 7.3.3

Depends R (>= 4.1.0)

Imports cograph (>= 2.3.6), ggplot2 (>= 3.4.0), grDevices, grid, stats, utils

Suggests testthat (>= 3.0.0), Nestimate, tna, knitr, rmarkdown

Config/testthat/edition 3

LazyData true

VignetteBuilder knitr

NeedsCompilation no

Author Mohammed Saqr [aut, cre, cph], Sonsoles López-Pernas [aut]

Maintainer Mohammed Saqr <saqr@saqr.me>

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

Date/Publication 2026-07-21 10:20:07 UTC

RemoteUrl <https://github.com/cran/lagdynamics>

RemoteRef HEAD

RemoteSha c2336c6daa3a1ddeb2f7d35548320f66853b13d7

Contents

ai_long	3
as.data.frame.lsa_comparison	4
as.data.frame.lsa_data	4
as.data.frame.lsa_reliability	5
bayes_compare_lsa	6
bootstrap_lsa	7
certainty_lsa	10
compare_lsa	11
engagement	14
get_lsa_engine	15
group_regulation	15
group_regulation_long	16
initial	17
lag_profile	18
list_lsa_engines	18
lsa_classical	19
lsa_data	22
lsa_ipf	24
lsa_lags	25
lsa_to_tna	26
lsa_transitions	27
nodes	28
permute_lsa	29
plot.lsa	30
plot.lsa_certainty	32
plot.lsa_comparison	32
plot_chords	34
plot_forest	35
plot_polar	36
plot_transitions	38
register_lsa_engine	39
reliability_lsa	40
stability_lsa	42
tests	43
transfer_entropy	44
transition_probabilities	45

<i>ai_long</i>	3
transitions	46
unregister_lsa_engine	49
Index	50

<i>ai_long</i>	<i>Human-AI Vibe Coding Interaction Events</i>
----------------	--

Description

A long-format event log of coded AI-side behaviours in human-AI vibe coding sessions. Each row is one coded AI event, with project/session identifiers, ordering variables, a fine-grained AI behaviour code, and a broader behaviour cluster. It is useful for demonstrating `lsa()`'s long-format event-log import path.

Usage

`ai_long`

Format

- A data.frame with 8,551 rows and 9 columns:
- message_id** Integer message identifier.
 - project** Project identifier.
 - session_id** Session identifier.
 - timestamp** Unix timestamp in seconds.
 - session_date** Session date as YYYY-MM-DD.
 - code** Fine-grained AI behaviour code.
 - cluster** Broader AI behaviour cluster.
 - code_order** Order of multiple codes within the same message.
 - order_in_session** Event order within session.

Details

The eight AI behaviour codes are Ask, Delegate, Execute, Explain, Investigate, Plan, Repair, and Report. The three clusters are Action, Communication, and Repair.

Source

Derived without modification from `Nestimate::ai_long` (Nestimate version 0.7.7, MIT license).

Examples

```
fit <- lsa(ai_long, session = "session_id",
          action = "code", order = "order_in_session")
transitions(fit, significant = TRUE)
```

```
as.data.frame.lsa_comparison
```

Tidy a Group Comparison

Description

Returns the per-edge comparison table (the same data frame as `x$edges`) so a comparison can be read with `as.data.frame()` like the other result objects, without reaching into the object.

Usage

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

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

Arguments

`x` An `lsa_comparison` or `lsa_comparison_pairwise` object.
`row.names, optional, ...` Standard `as.data.frame()` arguments (unused; present for method consistency).

Value

The tidy per-edge data frame.

Examples

```
grp <- ifelse(group_regulation$T1 == "plan", "plan", "other")
g <- lsa(group_regulation, group = grp)
cmp <- compare_lsa(g, R = 200)
head(as.data.frame(cmp))
```

```
as.data.frame.lsa_data
```

Tidy the Canonical Sequence Object

Description

Returns the canonical `lsa_data` as a tidy data frame: one row per event (`seq_id`, within-sequence index, state) for event-level input, or one row per from/to/count cell for transition-matrix input.

Usage

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

Arguments

`x` An lsa_data object from `lsa_data()`.
`row.names, optional, ...` Standard `as.data.frame()` arguments (unused; present for method consistency).

Value

A tidy data frame.

Examples

```
d <- lsa_data(c("a", "b", "a", "c", "b"))
as.data.frame(d)
```

```
as.data.frame.lsa_reliability
```

Tidy the per-replicate split-half correlations

Description

Tidy the per-replicate split-half correlations

Usage

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

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

Arguments

`x` An lsa_reliability (or lsa_reliability_group) object.
`row.names, optional, ...` Ignored (method signature compatibility).

Value

A data.frame, one row per replicate, with columns replicate and correlation (a grouped object gains a leading group column). NA correlations from degenerate splits are kept.

Examples

```
fit <- lsa(group_regulation)
rel <- reliability_lsa(fit, R = 200)
head(as.data.frame(rel))
```

bayes_compare_lsa	<i>Bayesian Comparison of Group Transition Structures (Dirichlet-Multinomial)</i>
-------------------	---

Description

Closed-form Bayesian alternative to [compare_lsa\(\)](#) for comparing the transition structures of two (or, pairwise, more) groups. Each state's outgoing transitions are modelled as Dirichlet-Multinomial with a Jeffreys prior, so each transition probability is marginally Beta. The per-edge posterior mean difference $\text{prob_a} - \text{prob_b}$ is exact; a credible interval, the probability of direction pd , and a two-sided Bayesian p -equivalent $2 * (1 - \text{pd})$ come from a Monte Carlo draw on the Beta marginals.

Usage

```
bayes_compare_lsa(
  x,
  y = NULL,
  prior = 0.5,
  draws = 10000L,
  ci = 0.95,
  mean_threshold = 0.01,
  bound_threshold = 0.001,
  adjust = "none",
  seed = NULL
)
```

Arguments

x	An <code>lsa_group</code> (two or more groups), or a single <code>lsa</code> fit for the first group.
y	The second group's <code>lsa</code> fit when <code>x</code> is a single fit; otherwise <code>NULL</code> .
prior	Numeric > 0. Dirichlet prior concentration added to every cell. Default 0.5 (Jeffreys). Use 1 for a uniform (Laplace) prior.
draws	Integer. Monte Carlo posterior draws for the credible intervals. Default 10000.
ci	Numeric in (0, 1). Credible-interval mass. Default 0.95.
mean_threshold, bound_threshold	An edge is flagged credibly different only if its credible interval excludes zero, $ \text{posterior mean diff} $ exceeds <code>mean_threshold</code> (default 0.01), and the credible bound nearest zero exceeds <code>bound_threshold</code> (default 0.001). The thresholds guard against differences that are detectable but negligibly small.

adjust	Multiple-comparison correction applied to the two-sided Bayesian p across edges (and family-wide across pairs); any method of <code>stats::p.adjust()</code> . Default "none".
seed	Optional integer for reproducible credible intervals.

Details

This complements `compare_lsa()`: the permutation test asks whether a difference is more extreme than chance; the Bayesian comparison asks for the plausible range of the true difference and how precisely it is estimated. An edge whose source state is rarely visited gets a wide credible interval even when its row-normalised probability looks decisive.

The result carries class `c("lsa_bayes", "lsa_comparison")` (and the pairwise object `c("lsa_bayes_pairwise", "lsa_comparison_pairwise")`), so `plot()` (barrel / heatmap) and `as.data.frame()` work as for a permutation comparison.

Value

For two groups, class `c("lsa_bayes", "lsa_comparison", "list")` with an edges data frame (from, to, prob_a, prob_b, diff, ci_low, ci_high, pd, effect_size, p_value, p_adj, significant), the two fits, and the Bayesian settings. For more than two groups, an all-pairwise `c("lsa_bayes_pairwise", "lsa_comparison_pairwise", "list")`.

References

Johnston, L. & Jendoubi, T. (2026). How Delivery Mode Reshapes Resource Engagement: A Bayesian Differential Network Analysis. TNA Workshop 2026.

See Also

`compare_lsa()`, `certainty_lsa()`

Examples

```
g <- lsa(group_regulation,
        group = ifelse(group_regulation$T1 == "plan", "p", "o"))
bc <- bayes_compare_lsa(g, seed = 1)
head(as.data.frame(bc))
```

bootstrap_lsa

Bootstrap Confidence Intervals for an LSA Fit

Description

Non-parametric bootstrap for any LSA fit produced by `lsa()`. Resamples the underlying sequence data (whole sequences when more than one is available; geometric-block stationary bootstrap on events otherwise), refits the engine on each resample using the immutable recipe stored in `fit$params`, and aggregates per-edge statistics into a tidy data frame.

Usage

```
bootstrap_lsa(
  fit,
  R = 1000L,
  level = c("auto", "sequence", "event"),
  block_length = NULL,
  level_alpha = 0.95,
  indices = NULL,
  parallel = FALSE,
  n_cores = NULL,
  verbose = FALSE,
  ...
)
```

Arguments

<code>fit</code>	An lsa object returned by <code>lsa()</code> .
<code>R</code>	Integer. Number of bootstrap replicates. Default 1000.
<code>level</code>	Character. Resampling unit: "sequence" (resample whole sequences with replacement, default when fit has more than one sequence), "event" (stationary block bootstrap on the event stream, used automatically for single-sequence input), or "auto" (default; pick based on <code>fit\$data\$n_sequences</code>).
<code>block_length</code>	For event-level bootstrap, mean geometric block length. Default <code>NULL</code> -> <code>ceiling(sqrt(T))</code> .
<code>level_alpha</code>	Numeric. Confidence level for percentile intervals. Default 0.95.
<code>indices</code>	Optional integer matrix of replay indices, one row per resample (row <code>b</code> is used for resample <code>b</code>). For sequence-level bootstrap it must be $R \times S$ with each entry a sequence index in $1 \dots S$. For event-level bootstrap it is $R \times T$ with each entry an event position in $1 \dots T$ (the fully expanded positions, not block starts). When supplied, replaces internal RNG and enables bit-identical reproducibility across runs. Dimensions and ranges are validated. See Details.
<code>parallel</code>	Logical. Use multi-core resampling. Default <code>FALSE</code> . Requires base R only (parallel package).
<code>n_cores</code>	Integer. Worker count when <code>parallel = TRUE</code> . Default <code>NULL</code> -> <code>parallel::detectCores() - 1</code> .
<code>verbose</code>	Logical. Print progress every 100 replicates. Default <code>FALSE</code> .
<code>...</code>	Reserved for future use.

Details

Sequence-level resampling (default for multi-sequence input). Each resample draws S sequence indices with replacement from `seq_len(S)` and rebuilds the multi-sequence input as the corresponding list of event vectors. Preserves within-sequence structure.

Event-level resampling (single-sequence input). Implements the stationary block bootstrap of Politis & Romano (1994). Block length is geometric with mean `block_length`; resampled blocks wrap around the event stream and are concatenated until total length equals the original T .

Reproducibility hook. Supply indices as an $R \times S$ integer matrix of sequence indices (sequence-level) or an $R \times T$ matrix of event positions (event-level) to deterministically replay the bootstrap across sessions, processes, or languages. The event-level matrix holds the fully expanded resampled positions, i.e. the same form produced internally, so a captured `indices_used` can be fed straight back in.

NA handling. Per-cell summary statistics (`mean`, `se`, `ci_low`, `ci_high`, `p_boot`) are computed with `na.rm = TRUE`, so replicates that produced NA for a given cell (for example structural-zero cells, or cells whose row marginal collapsed to zero in the resampled data) are excluded from that cell's summary. The summary therefore reflects only the finite replicates; cells whose every replicate was NA come back as NA themselves.

Value

An object of class `c("lsa_bootstrap", "list")` with:

edges Tidy per-edge data frame with observed + bootstrap mean, `se`, `ci_low`, `ci_high`, `p_boot`, and `stable` for count, `adj_res`, `prob`, and `yules_q`.

boot_obs $R \times K^2$ numeric matrix: cell-wise observed count from each replicate (flattened in `as.vector(obs)` order).

boot_adj_res $R \times K^2$ matrix of adjusted residuals.

R, level, level_alpha, indices_used Recipe metadata.

fit Reference to the original fit (for `$params / labels`).

References

Efron, B. (1979). Bootstrap methods: another look at the jackknife. *Annals of Statistics*, 7(1), 1-26.

Politis, D. N., & Romano, J. P. (1994). The stationary bootstrap. *Journal of the American Statistical Association*, 89(428), 1303-1313.

See Also

[permute_lsa\(\)](#), [stability_lsa\(\)](#)

Examples

```
fit <- lsa(engagement, engine = "classical")
bs <- bootstrap_lsa(fit, R = 200)
head(bs$edges)
```

certainty_lsa

*Analytic Certainty of Transition Edges (Dirichlet-Multinomial)***Description**

Closed-form Bayesian alternative to `bootstrap_lsa()` for the transition-probability edges of an `lsa` fit. Each state's outgoing transitions are modelled as Dirichlet-Multinomial: with a Jeffreys prior the posterior for a row is `Dirichlet(count + prior)`, so each edge probability is marginally `Beta(a, b)` and its posterior mean, standard deviation, credible interval and stability decision are available analytically. No resampling, so it runs in microseconds.

Usage

```
certainty_lsa(
  fit,
  prior = 0.5,
  level_alpha = 0.95,
  inference = c("stability", "threshold"),
  consistency_range = c(0.75, 1.25),
  edge_threshold = NULL
)
```

Arguments

<code>fit</code>	An <code>lsa</code> fit from <code>lsa()</code> , or an <code>lsa_group</code> .
<code>prior</code>	Numeric > 0. Dirichlet prior concentration added to every cell. Default 0.5 (the Jeffreys prior).
<code>level_alpha</code>	Numeric in (0, 1). Credible-interval level. Default 0.95 (a 95% interval), matching <code>bootstrap_lsa()</code> .
<code>inference</code>	"stability" (default) flags an edge whose posterior keeps it within a multiplicative <code>consistency_range</code> of its observed probability; "threshold" flags an edge whose posterior mass lies above <code>edge_threshold</code> .
<code>consistency_range</code>	Length-2 multiplicative bounds for stability inference. Default <code>c(0.75, 1.25)</code> .
<code>edge_threshold</code>	Numeric or NULL. Fixed threshold for <code>inference = "threshold"</code> ; NULL uses the 0.10 quantile of non-zero edge probabilities.

Details

The result carries class `c("lsa_certainty", "lsa_bootstrap")` and an edges table with the columns `plot_forest()` and `as.data.frame()` expect, so it is a drop-in for a bootstrap result (use `metric = "prob"`).

Certainty vs bootstrap. Both answer "how precisely is this edge pinned down?". They agree on homogeneous data. The Dirichlet posterior treats transitions as independent, so on strongly heterogeneous data (a mixture of latent classes with long sequences) it reports *more* certainty than the sequence bootstrap – prefer `bootstrap_lsa()` then.

Value

An object of class `c("lsa_certainty", "lsa_bootstrap", "list")` with an edges data frame (from, to, prob_observed, prob_mean, prob_se, prob_ci_low, prob_ci_high, p_value, stable, plus adj_res_observed/adj_res_stable for plotting), the posterior matrices (mean, sd, ci_lower, ci_upper), and call metadata (prior, level_alpha, inference, ...). For an lsa_group, a named list of these (class lsa_certainty_group).

References

Johnston, L. & Jendoubi, T. (2026). How Delivery Mode Reshapes Resource Engagement: A Bayesian Differential Network Analysis. TNA Workshop 2026.

See Also

[bootstrap_lsa\(\)](#), [stability_lsa\(\)](#), [plot_forest\(\)](#)

Examples

```
fit <- lsa(engagement)
cert <- certainty_lsa(fit)
cert
head(as.data.frame(cert))
```

compare_lsa

Compare Groups' Transition Structures

Description

Permutation test for whether groups produce different LSA transition structures. For each pair of groups it pools their sequences, repeatedly reassigns the group label of whole sequences (preserving the original group sizes), refits each pseudo-group, and builds a permutation distribution of the per-edge difference in the chosen measure. The two-sided p-value is $(1 + \#\{ |diff_perm| \geq |diff_obs| \}) / (1 + R)$ (Phipson & Smyth 2010). A single omnibus test of overall difference is reported from the same permutations.

Usage

```
compare_lsa(
  x,
  y = NULL,
  R = 1000L,
  measure = c("log_or", "adj_res", "yules_q", "prob", "count", "lift"),
  adjust = "none",
  min_count = 5L,
  parallel = FALSE,
```

```

    n_cores = NULL,
    verbose = FALSE,
    ...
)

```

Arguments

x	Either an <code>lsa_group</code> object (from <code>lsa(..., group =)</code>) with two or more groups, or a single <code>lsa</code> fit for the first group.
y	When x is a single <code>lsa</code> fit, the second group's <code>lsa</code> fit. Ignored (and must be <code>NULL</code>) when x is an <code>lsa_group</code> .
R	Integer. Number of label permutations per comparison. Default 1000.
measure	Character. The per-edge quantity compared between groups. Default "log_or": the per-cell log odds ratio of the 2x2 transition collapse (Haldane-Anscombe corrected on empty cells) – an N-invariant LSA effect size, so the comparison reflects behaviour rather than sample size. Other options: "yules_q" (also N-invariant, but saturates at +/-1 on zero cells), "adj_res" (adjusted residuals – the LSA <i>test statistic</i> , which scales with $\sqrt{N}$ and is therefore confounded by group size; a message warns when groups differ in size), "prob" (transition probabilities – a raw rate, i.e. the TNA quantity, with no independence baseline), "count", or "lift" (observed / expected).
adjust	Multiple-comparison correction; any method accepted by <code>stats::p.adjust()</code> (e.g. "holm", "BH", "bonferroni"). Default "none". For more than two groups it is applied across the pooled per-edge p-values of all pairs.
min_count	Integer. Minimum pooled observed count (group a + group b) for a transition to be tested. Default 5. Rarer cells carry an unstable odds ratio and a near-degenerate permutation null that produces spurious small p-values, so they get <code>p = NA</code> and are excluded from the multiple-comparison family and the omnibus rather than flagged significant. Set 0 to test every cell.
parallel	Logical. Use multi-core resampling. Default FALSE.
n_cores	Integer. Worker count when <code>parallel = TRUE</code> .
verbose	Logical. Print progress every 100 permutations.
...	Reserved.

NA handling. Non-estimable cells (structural zeros, zero-margin rows in a permuted pseudo-group) carry NA in the measure matrix and are never coerced to zero. Such cells get `p_perm = NA` rather than a spurious significant flag, and the exceedance tally and omnibus statistic are computed with `na.rm = TRUE`, matching `permute_lsa()`.

Interpretation caveats. The odds ratio is non-collapsible: the per-group log odds ratios are group-specific departure-from- independence measures and should not be pooled across groups that have different marginal state distributions. As with any LSA, a between-group difference can also be driven by subgroup composition (Simpson's paradox); confirm with subgroup analysis when a confound is plausible.

Details

With exactly two groups a single comparison is returned. With more than two groups every pairwise comparison is run and the requested adjust correction is applied **once across the whole family** of per-edge p-values (and separately across the per-pair omnibus tests), giving family-wise control rather than per-pair control.

Value

For two groups, an object of class `c("lsa_comparison", "list")` with:

edges Tidy per-edge data frame: from, to, the measure in each group (`<measure>_a`, `<measure>_b`), their difference `diff` (`= a - b`), the permutation p-value `p_perm`, the adjusted p-value `p_adj`, and a significant flag.

global Omnibus test list: statistic (observed sum of squared edge differences), `p_value`, and `R`.

perm_diff $R \times K^2$ matrix of permuted edge differences.

measure, R, adjust, groups Call metadata; `groups` is the length-two character vector of group labels (`a`, `b`).

fits The two original fits, named by group.

For more than two groups, an object of class `c("lsa_comparison_pairwise", "list")` with:

edges Tidy per-edge data frame across all pairs, prefixed by `group_a`, `group_b`; `p_adj` and significant reflect the family-wide correction.

global One row per pair: `group_a`, `group_b`, statistic, `p_value`, and the across-pairs adjusted `p_adj`.

comparisons Named list of the underlying two-group `lsa_comparison` objects (each fit with `adjust = "none"`), for drill-down.

measure, R, adjust, groups Call metadata; `groups` lists all group labels.

References

Phipson, B., & Smyth, G. K. (2010). Permutation p-values should never be zero. *Statistical Applications in Genetics and Molecular Biology*, 9(1), Article 39.

van Borkulo, C. D., et al. (2022). Comparing network structures on three aspects: A permutation test. *Psychological Methods*.

See Also

[permute_lsa\(\)](#), [bootstrap_lsa\(\)](#)

Examples

```
# group_regulation is wide sequences with no grouping column, so
# derive one: sessions whose first regulation act is planning vs not.
grp <- ifelse(group_regulation$T1 == "plan", "starts_plan", "other")
g <- lsa(group_regulation, group = grp)
cmp <- compare_lsa(g, R = 200)
head(cmp$edges)
```

```
cmp$global
```

engagement

Student Engagement Trajectories

Description

Wide-format categorical sequence data: 138 students observed over 15 weekly time points. Each row is one student; each column is a week. Entries are the student's engagement state for that week, one of "Active", "Average", "Disengaged", or NA for missing weeks.

Usage

```
engagement
```

Format

A data frame with 138 rows and 15 character columns.

Details

This is a standard small-K, multi-sequence example for lag sequential analysis: $K = 3$ states, $S = 138$ sequences, mean sequence length about 15. It exercises the wide-data input path of `lsa_data()` and produces a stable transition pattern with clear adjusted-residual signals.

Source

Derived from the trajectories matrix in the Nestimate package (<https://github.com/mohsaqr/Nestimate>), which is MIT-licensed and produced by Saqr and collaborators as a synthetic engagement trajectory example. The state values are preserved and stored here as a data frame for compatibility with sequence-data tooling.

Examples

```
fit <- lsa(engagement, engine = "classical")
fit
fit$adj_res
```

get_lsa_engine	<i>Retrieve a Registered LSA Engine</i>
----------------	---

Description

Retrieve a Registered LSA Engine

Usage

```
get_lsa_engine(name)
```

Arguments

name	Character scalar. The engine's identifier.
------	--

Value

The registry entry: a list with elements name, fn, description, requires.

See Also

[register_lsa_engine\(\)](#), [list_lsa_engines\(\)](#)

Examples

```
classical <- get_lsa_engine("classical")
classical$description
```

group_regulation	<i>Collaborative Learning Self-Regulation Sequences</i>
------------------	---

Description

A wide-format dataset of group regulation during collaborative learning: each row is one group's session, each column an ordered time point, and each cell a coded regulation action. Shorter sessions are padded with NA. It is the flagship example for the package vignette: 9 states over 2,000 sequences gives a realistically rich transition network.

Usage

```
group_regulation
```

Format

A data.frame with 2,000 rows (sequences) and 26 columns (ordered time points), character-coded.

Details

The nine coded actions are adapt, cohesion, consensus, coregulate, discuss, emotion, monitor, plan, and synthesis.

Examples

```
fit <- lsa(group_regulation)
transitions(fit, significant = TRUE)
```

group_regulation_long *Group Regulation Long Event Log*

Description

A long-format event log of collaborative-learning regulation actions, one row per coded event. It is the long companion to the wide [group_regulation](#) sequence matrix: the same nine regulation actions, recorded here with actor, session, timing, and grouping columns so the package's long-format import, grouping, and between-group comparison paths can be demonstrated on a single realistic data set.

Usage

```
group_regulation_long
```

Format

A data.frame with 27,533 rows and 6 columns:

Actor Integer student identifier (2,000 students).

Achiever Achievement group, High or Low.

Group Numeric collaboration-group identifier.

Course Course identifier (A, B, or C).

Time Event timestamp (POSIXct).

Action The regulation action code.

Details

The nine actions are adapt, cohesion, consensus, coregulate, discuss, emotion, monitor, plan, and synthesis. The Achiever column (High / Low) is a recorded grouping variable suitable for between-group analysis.

Source

Derived without modification from `tna::group_regulation_long` (tna package, MIT license).

See Also

[group_regulation](#) (the wide sequence matrix of the same actions)

Examples

```
fit <- lsa(group_regulation_long, actor = "Actor",
          action = "Action", time = "Time")
transitions(fit, significant = TRUE)
```

initial

Initial-State Distribution of an LSA Fit (Tidy)

Description

The proportion of sequences starting in each state, as a tidy data.frame. These are the initial-state probabilities (init P) that complement the transition-probability matrix from [transition_probabilities\(\)](#).

Usage

```
initial(fit)

## S3 method for class 'lsa'
initial(fit)

## S3 method for class 'lsa_group'
initial(fit)
```

Arguments

fit An lsa fit from [lsa\(\)](#).

Value

A data.frame with columns state, init_prob; zero rows when the fit came from a transition matrix (no initial states).

See Also

[transitions\(\)](#), [nodes\(\)](#)

Examples

```
initial(lsa(group_regulation))
```

lag_profile	<i>Lag Profile of a Single Transition</i>
-------------	---

Description

How one from -> to transition behaves across lags, as a tidy one-row-per-lag data frame. A clean shortcut for "track this transition over lags 1, 2, 3, ...".

Usage

```
lag_profile(x, from, to, lags = 1:3, ...)
```

Arguments

x	Sequence input (any form accepted by lsa()) or an existing lsa_lags() object.
from, to	State labels of the transition to profile.
lags	Integer vector of lags. Default 1:3. Ignored when x is already an lsa_lags object.
...	Passed to lsa_lags() when x is raw data.

Value

A tidy data.frame, one row per lag, with columns lag, from, to, count, prob, adj_res, p, and significant.

See Also

[lsa_lags\(\)](#)

Examples

```
lag_profile(group_regulation, "plan", "consensus", lags = 1:3)
```

list_lsa_engines	<i>List All Registered LSA Engines</i>
------------------	--

Description

List All Registered LSA Engines

Usage

```
list_lsa_engines()
```

Value

A data.frame with columns name, description, requires.

See Also

[register_lsa_engine\(\)](#), [get_lsa_engine\(\)](#)

Examples

```
list_lsa_engines()
```

lsa_classical	<i>Lag Sequential Analysis</i>
---------------	--------------------------------

Description

Fits a lag sequential analysis (LSA) on categorical event sequence data using a registered engine. Returns a tidy S3 object with named slots for observed/expected/probability/residual matrices and a long-format edge table suitable for transition-network visualization.

Usage

```
lsa_classical(data, ...)
```

```
lsa_two_cell(data, ...)
```

```
lsa_bidirectional(data, ...)
```

```
lsa_parallel_dominance(data, ...)
```

```
lsa_nonparallel_dominance(data, ...)
```

```
lsa(
  data,
  lag = 1,
  engine = "classical",
  alternative = c("two.sided", "greater", "less"),
  alpha = 0.05,
  loops = TRUE,
  structural_zeros = NULL,
  labels = NULL,
  group = NULL,
  actor = NULL,
  action = NULL,
  time = NULL,
  order = NULL,
```

```

    session = NULL,
    time_threshold = 900,
    custom_format = NULL,
    is_unix_time = FALSE,
    unix_time_unit = "seconds",
    params = list(),
    ...
)

```

Arguments

data	Sequence input (any form accepted by <code>lsa_data()</code>), or a raw long-format event-log data.frame when the actor / action arguments are supplied (see below). Accepted already- sequenced forms include vectors, lists of sequences, wide matrices/data.frames, transition-count matrices, and sequence- bearing objects (tna, group_tna, nestimate_data, stslst). NA and empty-string cells are treated as missingness, not as a state: they are dropped wherever they occur and no transition is counted into or out of them. To model missingness as its own state, recode it (e.g. NA -> "missing") before calling <code>lsa()</code> .
...	Additional engine-specific parameters (merged into params).
lag	Integer. The transition lag. Default 1. Positive lags count successors (state at $t \rightarrow t + \text{lag}$); negative lags count predecessors (what occurred $ \text{lag} $ steps before); 0 pairs each event with itself (degenerate for single-stream event data – genuine co-occurrence needs concurrent codes, not yet supported). Pre-computed transition-matrix input supports $\text{lag} = 1$ only. To analyse several lags at once, see <code>lsa_lags()</code> .
engine	Character scalar. The engine name, registered via <code>register_lsa_engine()</code> . Built-in engines: "classical", "two_cell", "bidirectional", "parallel_dominance", "nonparallel_dominance". Default "classical".
alternative	Character scalar. The alternative hypothesis for adjusted-residual and kappa p-values: one of "two.sided" (default), "greater", or "less".
alpha	Numeric. Significance threshold used to mark edges as significant in <code>fit\$edges\$significant</code> . Default 0.05.
loops	Logical. Keep self-transitions (the diagonal)? Default TRUE . Set <code>loops = FALSE</code> to forbid every self-transition – the common reason to exclude cells – without building a matrix by hand.
structural_zeros	Optional $K \times K$ 0/1 matrix for an <i>arbitrary</i> forbidden-cell pattern, where 0 marks a forbidden (structural-zero) cell and 1 an estimable one. Default NULL: every cell is part of the model. Combines with <code>loops</code> : <code>loops = FALSE</code> also zeros the diagonal of a supplied matrix. When any cell is forbidden the engine switches to iterative proportional fitting and Christensen's design-matrix residuals (see <code>inst/REFERENCES.md</code> §2.2, §4.2).
labels	Optional character vector of state labels.
group	Optional grouping for a multi-group fit. Either a vector with one entry per input sequence (length <code>n_sequences</code>), or — for long-format input (see actor/action)

— the **name of a grouping column** in the log, which must be constant within each actor/session so each recovered sequence maps to one group. Sequences are partitioned by group and a separate lsa fit is built for each. All group fits share one global label set (derived from the full data) so their $K \times K$ matrices are directly comparable, even when a group never visits some state. Returns an lsa_group object (a named list of lsa fits). Requires event-level input; a pre-computed transition matrix cannot be split by group. Default NULL (single-group fit).

actor, action, time, order, session	Column names (each a single string) for long-format event-log input. Supplying action switches lsa() into long-format mode: the raw log in data is sequenced by grouping rows per actor, per session, per actor $\times$ session, or as one global sequence when neither boundary is supplied. Events are ordered within each sequence by order if given else by time, and – when time is given and no session column is – a new sequence starts whenever the gap between consecutive events exceeds time_threshold seconds. All NULL by default except action in long-format mode (input is otherwise taken as already-sequenced).
time_threshold	Numeric. Maximum gap in seconds between consecutive events before a new session is started in long-format mode. Default 900 (15 minutes). Ignored unless time is given and session is not.
custom_format	Optional strptime format string for parsing the time column (e.g. "%Y-%m-%d %H:%M:%S"). Default NULL (native date/time classes and ISO strings are parsed directly).
is_unix_time	Logical. Treat the time column as a Unix epoch. Default FALSE.
unix_time_unit	Character. Unit of the Unix epoch when is_unix_time = TRUE: "seconds" (default), "milliseconds", or "microseconds".
params	Optional named list of engine-specific parameters forwarded to the engine function.

Value

An object of class c("lsa", "cograph_network"). Read it with the verbs rather than by reaching into slots: [transitions\(\)](#) for the tidy edge table, [nodes\(\)](#), [tests\(\)](#), [initial\(\)](#), and [summary\(\)](#) for the other results, and [plot\(\)](#)/[plot_transitions\(\)](#) to draw it. Every number a verb returns is backed by these slots:

- edges** The tidy one-row-per-transition frame that backs [transitions\(\)](#) (with extra cograph_network protocol columns).
- nodes** Data frame backing [nodes\(\)](#): id, label, name, outgoing, incoming.
- obs, exp, prob, prob_col, adj_res, p, yules_q, kappa, kappa_z, kappa_p** The same per-cell quantities as edges, in $K \times K$ matrix form (prob is row-conditional $P(\text{to} | \text{from})$, prob_col column-conditional $P(\text{from} | \text{to})$). Convenient for matrix algebra; not the primary interface.
- lrx2, x2** Lists (statistic, df, p) backing [tests\(\)](#): the tablewise likelihood-ratio (G^2) and Pearson chi-square tests of independence; NULL for engines without an expected table.
- inits** Named numeric vector backing [initial\(\)](#) (proportion of sequences starting in each state, sums to 1); NULL for transition-matrix input.

weights $K \times K$ matrix used as the default edge weight for plotting. Equal to obs (counts) by default.

directed Logical scalar; TRUE for directed engines, FALSE for bidirectional.

method Engine name (the slot the cograph_network protocol reads). Also recorded in params\$engine.

data The canonical lsa_data object (events + seq_id).

params Immutable snapshot of all parameters used (recipe), including params\$engine.

meta List with source, IPF info, version, and call.

When group is supplied, returns an object of class `c("lsa_group", "list")`: a named list of lsa fits (one per group level) carrying levels, group_sizes, labels, and engine attributes. Downstream verbs (`transitions()`, `transition_probabilities()`, `reliability_lsa()`, etc.) dispatch on it and return grouped results.

See Also

`lsa_data()`, `lsa_transitions()`, `register_lsa_engine()`, `list_lsa_engines()`

Examples

```
seq <- c("Question", "Explain", "Agree",
        "Question", "Explain", "Elaborate",
        "Agree", "Question", "Explain")
fit <- lsa(seq, engine = "classical")
fit
head(fit$edges)
```

lsa_data

Canonicalize Sequence Input for Lag Sequential Analysis

Description

Coerces a wide variety of user input shapes into a single canonical representation used by every downstream lagdynamics function (engines, bootstrap, permutation, grouping, plotting).

Usage

```
lsa_data(x, labels = NULL)
```

Arguments

x	Sequence input. See Details.
labels	Optional character vector of label names for the states. When NULL, labels are extracted from the data: unique sorted values of character input, or "Code 1", "Code 2", ... for integer input.

Details

Accepted input forms:

- An atomic vector of integer or character codes — treated as a single sequence.
- A list of atomic vectors — treated as multiple independent sequences; transitions are not counted across sequence boundaries.
- A wide matrix or data.frame with rows = sequences, columns = ordered time points. Missing values (NA) and empty strings are treated as missingness, not as a state: they are dropped wherever they occur in a row and the surrounding events close up, so no transition is counted into or out of a gap. To model missingness as its own state, recode it (e.g. NA -> "missing") before calling `lsa()`.
- A square numeric matrix of pre-computed transition counts. Row *i*, column *j* is the count of *i* -> *j* transitions. In this case events and seq_id are not available and downstream resampling tools that need event-level data will error.
- A sequence-bearing object: a tna or group_tna (sequences read from its \$data slot), a tna_data or nestimate_data (\$sequence_data), or an stslst. The stored event sequences are recovered and analysed. A tna built from a bare matrix (no retained sequences) errors, because transition *counts* cannot be recovered from probability weights.

Value

An object of class `c("lsa_data", "list")` with elements:

events Integer vector of event codes (1-indexed), or NULL if input was a transition matrix.

seq_id Integer vector of sequence membership, same length as events, or NULL if input was a transition matrix.

labels Character vector of state labels.

n_states Number of distinct states (K).

n_sequences Integer count of independent sequences.

n_events Total number of events across all sequences.

transitions_per_seq Integer vector: number of transitions each sequence contributes at lag 1.

source One of "events", "transitions" — flags whether event-level data is available.

obs_input If source = "transitions", the original K x K count matrix. Otherwise NULL.

See Also

`lsa_transitions()`, `lsa()`

Examples

```
# Single character sequence
d1 <- lsa_data(c("a", "b", "a", "c", "b"))
d1$n_events

# Multiple sequences
d2 <- lsa_data(list(c("a", "b", "a"), c("b", "c", "a", "b")))
```

```
d2$n_sequences

# Pre-computed transition matrix
tm <- matrix(c(0, 3, 1, 2, 0, 4, 5, 1, 0), 3, 3,
             dimnames = list(c("a", "b", "c"), c("a", "b", "c")))
d3 <- lsa_data(tm)
d3$source
```

lsa_ipf	<i>Iterative Proportional Fitting for Two-Way Tables with Structural Zeros</i>
---------	--

Description

Fits expected cell frequencies under the row + column independence model when some cells are constrained to be exactly zero (structural zeros). The fitted table has the same row and column marginals as obs, satisfies $E[i, j] = 0$ wherever $structure[i, j] = 0$, and is the maximum likelihood estimate under the independence model restricted to the non-zero pattern.

Usage

```
lsa_ipf(obs, structure = NULL, tol = 1e-08, max_iter = 200L)
```

Arguments

obs	Numeric $K \times K$ matrix of observed counts.
structure	Numeric $K \times K$ 0/1 matrix. Default <code>matrix(1, K, K)</code> : every cell, including the diagonal, is estimable – self-transitions and every observed cell are kept. A 0 marks a cell as a structural zero (forbidden), a 1 marks it as estimable. Pass an explicit pattern (e.g. <code>1 - diag(K)</code>) only when you want to <i>opt out</i> of specific cells because the coding scheme makes them impossible by construction.
tol	Numeric. Convergence tolerance on marginal differences. Default <code>1e-8</code> .
max_iter	Integer. Maximum number of row+column scaling passes. Default <code>200L</code> .

Details

Implementation follows Wickens (1989), pp. 107-112: alternately scale rows then columns of an initialized expected table until row and column marginals converge to those of obs within tol.

Value

A list with elements:

fit The fitted $K \times K$ expected-frequency matrix.

iterations Number of row+column passes used.

converged Logical scalar: whether convergence was reached within `max_iter`.

max_margin_diff Maximum absolute difference between observed and fitted marginals at termination.

References

Wickens, T. D. (1989). *Multiway contingency tables analysis for the social sciences*, pp. 107-112. Lawrence Erlbaum.

Examples

```
obs <- matrix(c(0, 4, 6,
               3, 0, 5,
               7, 2, 0), nrow = 3, byrow = TRUE)
fit <- lsa_ipf(obs)
fit$fit                                     # fitted expected counts
all.equal(rowSums(fit$fit), rowSums(obs))   # TRUE
all.equal(colSums(fit$fit), colSums(obs))   # TRUE
```

lsa_lags

Lag Sequential Analysis Across Several Lags

Description

Fits `lsa()` at each requested lag and returns the fits together, so you can compare a transition's strength across lags (a *lag profile*). Each element is an ordinary lsa fit.

Usage

```
lsa_lags(data, lags = 1:3, ...)
```

Arguments

data	Sequence input (any form accepted by <code>lsa()</code>).
lags	Integer vector of lags. May include negative lags (predecessors) and 0. Default 1:3.
...	Passed to <code>lsa()</code> (e.g. engine, alpha, structural_zeros).

Value

An object of class `c("lsa_lags", "list")`: a named list of lsa fits (names "lag1", "lag2", ...), with a lags attribute. `as.data.frame()` on it row-binds `transitions()` of every fit (each already carries its lag column) into one tidy long frame with the same columns as `transitions()`.

See Also

[lsa\(\)](#)

Examples

```
prof <- lsa_lags(engagement, lags = 1:3)
prof
# Track one transition across lags with the dedicated verb:
lag_profile(engagement, from = "Active", to = "Average", lags = 1:3)
```

lsa_to_tna

Convert an lsa Fit to a tna Network

Description

Convert an lsa fit to a tna-class network object usable by the tna package's centrality, pruning, community, and bootstrap routines.

Usage

```
lsa_to_tna(x, ...)

## S3 method for class 'lsa'
lsa_to_tna(x, weights = c("prob", "count", "adj_res", "lift"), ...)

## S3 method for class 'lsa_group'
lsa_to_tna(x, weights = c("prob", "count", "adj_res", "lift"), ...)
```

Arguments

x	An lsa fit from <code>lsa()</code> , or an lsa_group from <code>lsa(..., group =)</code> .
...	Method-specific arguments.
weights	Character. Which matrix to expose as the tna edge weights. One of "prob" (row-normalised probabilities, default), "count" (raw observed counts), "adj_res" (adjusted residuals, over-representation network), or "lift" (observed / expected association strength).

Details

This function is deliberately named `lsa_to_tna()`, not `as_tna()`, to avoid overlapping export names with sibling packages.

Value

For an lsa fit, a tna object with weights, inits, labels, and, when available, data slots. For an lsa_group, a group_tna object.

Examples

```
fit <- lsa(engagement)
net <- lsa_to_tna(fit, weights = "prob")
tna::centralities(net)
```

lsa_transitions

*Tidy Transition Counts at a Given Lag***Description**

Computes the $K \times K$ transition count matrix from canonical lag sequential data, optionally returning a tidy long-format edge table alongside the matrix.

Usage

```
lsa_transitions(x, lag = 1)
```

Arguments

x	Either an lsa_data object or any input accepted by lsa_data() (which will be coerced).
lag	Integer. The lag at which to count transitions; default 1. A positive lag counts successors (from at t , to at $t + \text{lag}$), a negative lag counts predecessors, and 0 pairs each event with itself (a degenerate diagonal). Must be a single finite whole number.

Details

Transitions are counted within sequences only; no transition spans a sequence boundary. For input that was supplied as a pre-computed transition matrix (`source = "transitions"` on the `lsa_data` object), the input matrix is returned at lag 1 and an error is raised for any other lag.

Value

An object of class `c("lsa_transitions", "list")` with elements:

obs The $K \times K$ observed transition count matrix with `dimnames` set to the labels.

row_totals Length- K vector `rowSums(obs)`.

col_totals Length- K vector `colSums(obs)`.

n_transitions Scalar `sum(obs)`.

lag The lag used.

labels Character vector of state labels.

edges Tidy long-format data.frame with one row per (from, to) cell containing columns `from`, `to`, `lag`, `count`, `row_total`, `col_total`, `n_transitions`.

See Also

[lsa_data\(\)](#), [lsa\(\)](#)

Examples

```
d <- lsa_data(c("a", "b", "a", "c", "b", "a"))
tx <- lsa_transitions(d, lag = 1)
tx$obs
head(tx$edges)
```

nodes

Nodes of an LSA Fit (Tidy)

Description

The states and their incoming / outgoing transition totals, as a tidy `data.frame` (one row per state).

Usage

```
nodes(fit)

## S3 method for class 'lsa'
nodes(fit)

## S3 method for class 'lsa_group'
nodes(fit)
```

Arguments

`fit` An `lsa` fit from [lsa\(\)](#).

Value

A `data.frame` with columns `state` (the state name, matching the `from/to` endpoints of [transitions\(\)](#)), `outgoing`, and `incoming` (its total out- and in-transition counts).

See Also

[transitions\(\)](#), [tests\(\)](#)

Examples

```
nodes(lsa(group_regulation))
```

permute_lsa

*Permutation Test for an LSA Fit***Description**

Empirical null-distribution p-values for every cell of an LSA transition matrix. Repeatedly shuffles the input event vector (within sequence boundaries) and recomputes the engine's residual matrix, producing a permutation distribution for each cell. The two-sided p-value is $(1 + \#\{ |\text{stat_perm}| \geq |\text{stat_obs}| \}) / (1 + \dots)$ (Phipson & Smyth 2010).

Usage

```
permute_lsa(
  fit,
  R = 1000L,
  within_sequence = TRUE,
  shuffles = NULL,
  parallel = FALSE,
  n_cores = NULL,
  verbose = FALSE,
  ...
)
```

Arguments

<code>fit</code>	An lsa object returned by <code>lsa()</code> .
<code>R</code>	Integer. Number of permutations. Default 1000.
<code>within_sequence</code>	Logical. When TRUE (default for multi-sequence input), each sequence is shuffled independently; when FALSE, the whole event stream is shuffled across sequence boundaries.
<code>shuffles</code>	Optional list of length R, each element an integer permutation of <code>seq_len(n_events)</code> . When supplied, replaces internal RNG.
<code>parallel</code>	Logical. Use multi-core resampling. Default FALSE.
<code>n_cores</code>	Integer. Worker count when <code>parallel = TRUE</code> .
<code>verbose</code>	Logical. Print progress every 100 replicates.
<code>...</code>	Reserved.

NA handling. The exceedance count that drives `p_perm` is computed with `na.rm = TRUE`, so replicates that produced NA for a cell (structural-zero cells, zero-margin cells in the permuted table) are excluded from that cell's tally rather than counted as either an exceedance or a non-exceedance.

Value

An object of class `c("lsa_permutation", "list")` with:

edges Tidy per-edge data frame with observed counts and residuals, the empirical permutation p-value `p_perm`, and a significant flag at the recipe's alpha threshold.

perm_adj_res $R \times K^2$ numeric matrix of permuted residuals (cells in `as.vector(adj_res)` order).

R, within_sequence Recipe metadata.

fit Reference to the original fit.

References

Castellan, N. J. (1992). Shuffling arrays: appearances may be deceiving. *Behavior Research Methods, Instruments, & Computers*, 24(1), 72-77.

Phipson, B., & Smyth, G. K. (2010). Permutation p-values should never be zero: calculating exact p-values when permutations are randomly drawn. *Statistical Applications in Genetics and Molecular Biology*, 9(1), Article 39.

See Also

[bootstrap_lsa\(\)](#), [stability_lsa\(\)](#)

Examples

```
fit <- lsa(engagement, engine = "classical")
pm <- permute_lsa(fit, R = 200)
head(pm$edges)
```

plot.lsa

Plot an LSA Fit

Description

One entry point for every view of a fit; pick it with type: "heatmap" (default, the from `x` to residual heatmap), "network" (transition network via [cograph::splot\(\)](#)), "chord" (chord diagram via [cograph::plot_chord\(\)](#)), or "sunburst" (polar rose). Extra arguments are forwarded to the chosen view's worker ([plot_transitions\(\)](#), [plot_chords\(\)](#), [plot_polar\(\)](#)); see those for view-specific options.

Usage

```
## S3 method for class 'lsa'
plot(x, type = c("heatmap", "network", "chord", "sunburst"), ...)

## S3 method for class 'lsa_group'
plot(
  x,
  type = c("heatmap", "network", "chord", "sunburst"),
  combined = FALSE,
  ...
)
```

Arguments

<code>x</code>	An lsa fit from <code>lsa()</code> .
<code>type</code>	Which view to draw: "heatmap" (default), "network", "chord", or "sunburst".
<code>...</code>	Forwarded to the chosen view. For "heatmap": which ("residuals" (default), "prob", "count", "expected"). For "network": weights ("tna" for the probability-weighted TNA network). For "chord": width, color. For "sunburst": style, fill.
<code>combined</code>	Logical, for a grouped fit only. FALSE (default) draws each group as its own full-size figure; TRUE tiles all groups into a single figure (compact, but cramped for many groups).

Value

A ggplot object for "heatmap" and "sunburst"; the (invisible) cograph object for "network" and "chord". Drawn when printed.

See Also

`plot_transitions()`, `plot_chords()`, `plot_polar()`, `plot_forest()`, `transitions()`

Examples

```
fit <- lsa(group_regulation)
plot(fit) # residual heatmap (default)
plot(fit, which = "prob") # heatmap of probabilities
plot(fit, type = "network") # transition network
plot(fit, type = "network", weights = "tna") # TNA probability network
plot(fit, type = "chord") # chord diagram
plot(fit, type = "sunburst") # polar sunburst
```

plot.lsa_certainty	<i>Plot an Analytic-Certainty Result</i>
--------------------	--

Description

Circular forest of the per-edge transition-probability credible intervals from `certainty_lsa()` (delegates to `plot_forest()` with `metric = "prob"`).

Usage

```
## S3 method for class 'lsa_certainty'
plot(x, metric = "prob", ...)
```

Arguments

<code>x</code>	An <code>lsa_certainty</code> object.
<code>metric</code>	Which credible interval to draw. Default <code>"prob"</code> .
<code>...</code>	Passed to <code>plot_forest()</code> .

Value

A ggplot object (drawn when printed). Needs ggplot2.

Examples

```
fit <- lsa(group_regulation)
cert <- certainty_lsa(fit)
plot(cert)
```

plot.lsa_comparison	<i>Plot a Group Comparison</i>
---------------------	--------------------------------

Description

Two views of a `compare_lsa()` result. The default `"barrel"` is a back-to-back pyramid (one row per transition): the first group's bar runs left and the second's right, bar length is each group's transition probability, bar colour is each group's log odds ratio (blue = over-represented, red = avoided, following the `vcd` / `mosaic` convention), bar ends show the observed count, and the centre chip shows the difference p-value (bold and starred when significant). The bar of the group with the higher value gets a border that darkens with the size of the difference (faint = small, dark = large). For more than two groups, one barrel is drawn per pair via facets. The `"heatmap"` style draws the signed difference as a from `x` to grid on the same diverging scale.

Usage

```
## S3 method for class 'lsa_comparison'
plot(
  x,
  style = c("barrel", "heatmap"),
  value = c("prob", "count"),
  rank = c("frequency", "effect"),
  top_n = 12L,
  ...
)

## S3 method for class 'lsa_comparison_pairwise'
plot(
  x,
  style = c("barrel", "heatmap"),
  value = c("prob", "count"),
  rank = c("frequency", "effect"),
  top_n = 12L,
  ...
)
```

Arguments

<code>x</code>	An <code>lsa_comparison</code> or <code>lsa_comparison_pairwise</code> object.
<code>style</code>	"barrel" (default) or "heatmap".
<code>value</code>	For "barrel", the quantity mapped to bar length: "prob" (transition probability, default) or "count".
<code>rank</code>	For "barrel", how to choose which transitions to show: "frequency" (default) ranks by pooled observed count – the backbone transitions, which are mostly over-represented (blue); "effect" ranks by the strongest association in either group ($ \log OR $, among tested cells), surfacing both over- (blue) and under-represented / avoided (red) transitions.
<code>top_n</code>	For "barrel", how many transitions to show (highest rank on top; for the pairwise object the ranking is shared across facets so they line up). Default 12.
<code>...</code>	Reserved.

Value

A ggplot object (drawn when printed). Needs ggplot2.

See Also

[compare_lsa\(\)](#), [plot.lsa\(\)](#)

Examples

```
grp <- ifelse(group_regulation$T1 == "plan", "starts_plan", "other")
```

```

g <- lsa(group_regulation, group = grp)
cmp <- compare_lsa(g, R = 200)
plot(cmp)                # back-to-back barrel
plot(cmp, style = "heatmap") # difference heatmap

```

plot_chords

Circular (Chord) Diagram of an LSA Fit

Description

Draws the transition structure as a chord diagram via `cograph::plot_chord()`: states are arcs on an outer ring and each transition is a curved ribbon whose **width** is its frequency (or probability) and whose **fill colour** is its adjusted residual (warm = over-represented, cool = avoided). Supply a second fit as `compare` to fill each ribbon by the *difference* in the colour metric between the two fits.

Usage

```

plot_chords(
  fit,
  compare = NULL,
  width = c("count", "prob"),
  color = c("residuals", "lift", "prob", "count"),
  significant = FALSE,
  self_loops = TRUE,
  alpha = 0.6,
  ...
)

```

Arguments

<code>fit</code>	An lsa fit from <code>lsa()</code> .
<code>compare</code>	Optional second lsa fit. When supplied, ribbon colour is <code>colour(fit) - colour(compare)</code> (a signed difference on the diverging scale). The two fits must share the same states. Default NULL.
<code>width</code>	Which non-negative quantity sets ribbon width: "count" (default, transition frequency) or "prob" (row-conditional probability).
<code>color</code>	Which quantity fills the ribbons: "residuals" (default, signed adjusted residual, diverging), "lift", "prob", or "count". Non-residual metrics use a sequential scale unless <code>compare</code> makes them a signed difference.
<code>significant</code>	Logical. Keep only significant transitions (drops the others' ribbons). Ignored when <code>compare</code> is set. Default FALSE.
<code>self_loops</code>	Logical. Draw self-transition ribbons. Default TRUE.
<code>alpha</code>	Ribbon fill opacity. Default 0.6.
<code>...</code>	Passed to <code>cograph::plot_chord()</code> (e.g. <code>ticks</code> , <code>segment_width</code> , <code>label_size</code> , <code>title</code>).

Details

This is the circular companion to the `plot.lsa()` heatmap and the `plot_transitions()` network. Like them it delegates the drawing to `cograph`; it needs the `cograph` package installed.

Value

Invisibly, the list returned by `cograph::plot_chord()` (segments and chords data frames). Drawn as a side effect.

See Also

`plot.lsa()` (heatmap), `plot_transitions()` (network), `transitions()`

Examples

```
fit <- lsa(group_regulation)
plot_chords(fit)                # ribbons filled by residual
plot_chords(fit, width = "prob") # width = probability
plot_chords(fit, significant = TRUE, ticks = TRUE)

# Compare two groups: ribbon colour = difference in residuals.
g <- lsa(group_regulation,
         group = rep(c("A", "B"), length.out = nrow(group_regulation)))
plot_chords(g$A, compare = g$B)
```

plot_forest

Circular Bootstrap Forest of an LSA Fit

Description

Draws a radial forest of an `bootstrap_lsa()` result: each transition is a spoke around a ring, spanning its bootstrap confidence interval, with a square at the observed estimate and a dashed reference ring at the null. Spokes whose adjusted residual is significant across resamples are coloured by direction (warm = over-represented, cool = avoided); non-significant ones are grey. Needs `ggplot2`.

Usage

```
plot_forest(
  boot,
  metric = c("residuals", "count", "prob", "yules_q"),
  n_top = NULL,
  show_nonsig = TRUE,
  label_size = 2.6
)

## S3 method for class 'lsa_bootstrap'
plot(x, ...)
```

Arguments

boot	An lsa_bootstrap object from <code>bootstrap_lsa()</code> .
metric	Which bootstrapped quantity to plot: "residuals" (default, adjusted residual), "count", "prob", or "yules_q".
n_top	Optional integer: keep only the n_top edges with the largest absolute estimate (the rest are dropped). Default NULL (all edges).
show_nonsig	Logical. Draw non-significant edges (grey). Default TRUE; set FALSE to keep only significant transitions.
label_size	Edge-label text size. Default 2.6.
x	An lsa_bootstrap object (for the <code>plot()</code> method).
...	Passed to <code>plot_forest()</code> (e.g. metric, n_top).

Value

A ggplot object (drawn when printed).

See Also

`bootstrap_lsa()`, `plot.lsa()` (heatmap), `plot_polar()` (sunburst)

Examples

```
fit <- lsa(group_regulation)
b <- bootstrap_lsa(fit, R = 500)
plot_forest(b) # residual CIs, circular
plot_forest(b, metric = "prob") # probability CIs
plot_forest(b, show_nonsig = FALSE) # significant transitions only
```

plot_polar

Polar Sunburst of an LSA Fit

Description

Draws the transition structure as a polar sunburst with the source states named along a large inner ring. Two styles: "rose" (default) gives every target an equal angular slot and encodes frequency as the radial **bar height** (so nothing crams); "wedge" sizes each transition's angular **width** by its frequency share (the classic look), omitting tiny wedges. Both fill by the adjusted residual (warm = over-represented, cool = avoided), sharing the `plot.lsa()` heatmap colour scale. Needs ggplot2.

Usage

```
plot_polar(
  fit,
  style = c("rose", "wedge"),
  fill = c("residuals", "prob", "lift"),
  size = c("count", "prob"),
  significant = FALSE,
  labels = c("all", "auto", "none"),
  min_show = 0.01,
  label_size = 3,
  ...
)
```

Arguments

<code>fit</code>	An lsa fit from <code>lsa()</code> .
<code>style</code>	"rose" (default, equal slots + bar height) or "wedge" (frequency-proportional wedge width).
<code>fill</code>	Which quantity fills the bars/wedges: "residuals" (default, diverging), "prob", or "lift".
<code>size</code>	For style = "rose", which non-negative quantity sets bar height: "count" (default) or "prob". Ignored for "wedge".
<code>significant</code>	Logical. Grey out non-significant cells (keeping their size). Default FALSE.
<code>labels</code>	Which target cells to name: "all", "auto", or "none". Default is "all" for "rose" (equal slots leave room) and "auto" for "wedge" (only wedges wide enough to fit a name). Source names are always shown.
<code>min_show</code>	For style = "wedge", drop wedges whose frequency share of their source's outflow is below this fraction. Default 0.01; 0 keeps all.
<code>label_size</code>	Label text size. Default 3.
<code>...</code>	Ignored; accepted so <code>plot(fit, type = "sunburst", ...)</code> can forward arguments without error.

Value

A ggplot object (drawn when printed).

See Also

`plot.lsa()` (heatmap), `plot_chords()` (chord), `plot_forest()` (bootstrap forest)

Examples

```
fit <- lsa(group_regulation)
plot_polar(fit) # rose: bars filled by residual
plot_polar(fit, style = "wedge") # classic frequency wedges
plot_polar(fit, significant = TRUE) # non-significant cells greyed
```

plot_transitions	<i>Plot the Transition Network</i>
------------------	------------------------------------

Description

Draws the directed transition network of an lsa fit with `cograph::splot()`. Pick the edge weight with `weights`; optionally keep only significant edges. Nodes are white and edges are labelled by default. Returns the cograph network invisibly.

Usage

```
plot_transitions(
  fit,
  weights = c("residuals", "tna", "relative", "count", "prob", "lift", "yules_q"),
  significant = FALSE,
  top = NULL,
  decimals = 1,
  node_fill = "white",
  edge_labels = TRUE,
  ...
)
```

Arguments

<code>fit</code>	An lsa fit from <code>lsa()</code> .
<code>weights</code>	Which matrix becomes the edge weight, and how it is drawn: "residuals" (default) – a residual network (not a transition one): adjusted residuals coloured by sign on the TNA / Nestimate convention, blue = more (over-represented) solid and red = less (avoided) dashed with a soft halo. "tna" / "relative" / "prob" / "count" – the familiar transition network of Transition Network Analysis (TNA), drawn with <code>cograph::splot(tna_styling = TRUE)</code>: <code>cograph</code>'s own TNA styling (coloured nodes, weighted directed edges) plus a donut ring per node carrying its initial-state probability, and edges labelled with the transition probability ("tna" / "relative" / "prob") or observed count ("count"). "tna" and "relative" are explicit aliases for "prob". For probability/TNA networks, edges below 0.05 are dropped by default so weak transitions do not clutter the plot (override with <code>edge_cutoff</code>). "lift" – observed / expected, drawn in a single neutral colour with magnitude carried by edge width. "yules_q" – a signed association network: Yule's Q on a fixed [-1, 1] scale, coloured by sign like the residual network (blue over-represented, red avoided) but bounded and not growing with sample size.
<code>significant</code>	Logical. Keep only edges whose adjusted-residual p-value is below the fit's alpha; weaker cells are set to 0 (no edge). Default FALSE. Note that at large

	sample sizes almost every cell is significant, so this is a weak visual filter – prefer top (effect-size pruning) to declutter a dense residual network.
top	Numeric or NULL. Keep only the strongest edges by absolute weight (applied after significant); the rest are set to 0. A fraction $0 < \text{top} < 1$ keeps that proportion of the present edges (top = 0.5 -> the strongest half); a value top ≥ 1 keeps that many edges (top = 12 -> the 12 strongest). The legible way to thin a dense residual network: it prunes by effect size (<code> adjusted_residual </code>) rather than by p-value. NULL (default) keeps every edge. Applies to every view; for the probability network it composes with the default <code>edge_cutoff = 0.05</code> .
decimals	Number of decimal places for edge labels. Default 1.
node_fill	Node fill colour. Default "white"; the probability / count networks use a per-state palette instead unless <code>node_fill</code> is set explicitly.
edge_labels	Logical (or a label vector). Show edge weights as labels. Default TRUE.
...	Passed to <code>cograph::splot()</code> (e.g. <code>node_shape</code> , <code>layout</code> , <code>edge_cutoff</code> , <code>curvature</code>).

Value

The `cograph_network` object, invisibly (drawn as a side effect).

See Also

`plot.lsa()` (heatmap), `transitions()`, `transition_probabilities()`

Examples

```
fit <- lsa(group_regulation)
plot_transitions(fit)                                # residual network
plot_transitions(fit, weights = "tna")               # TNA probabilities
plot_transitions(fit, weights = "relative")          # same as "tna"
plot_transitions(fit, weights = "prob")              # same matrix
plot_transitions(fit, weights = "residuals",         # residual network,
                 significant = TRUE)                 #   significant only
plot_transitions(fit, top = 12)                      # 12 strongest edges
plot_transitions(fit, top = 0.5)                    # strongest 50%
plot_transitions(fit, decimals = 2)                 # 2-dp edge labels
plot_transitions(fit, node_shape = "square")         # splot passthrough
```

register_lsa_engine	<i>Register a Lag Sequential Analysis Engine</i>
---------------------	--

Description

Adds a new engine to the lagdynamics registry so it can be referenced by name via `lsa(..., engine = "<name>")`. Built-in engines ("classical", "two_cell", "bidirectional", "parallel_dominance", "nonparallel_dominance") are registered automatically when the package loads.

Usage

```
register_lsa_engine(name, fn, description, requires = character())
```

Arguments

name	Character scalar. The engine's identifier as used in <code>lsa(engine = name)</code> .
fn	A function. Must accept a transitions argument (a tidy transition table produced by <code>lsa_transitions()</code>) and arbitrary named ... arguments forwarded from <code>lsa(params = list(...))</code> . Must return a named list with at least the matrix elements obs, exp, prob, adj_res, and p (each $K \times K$). Additional $K \times K$ matrices are preserved as engine-specific edge statistics.
description	Character scalar. One-line human-readable description shown by <code>list_lsa_engines()</code> .
requires	Character vector. Names of packages the engine depends on. Empty by default.

Value

Invisibly returns name.

See Also

`get_lsa_engine()`, `list_lsa_engines()`, `unregister_lsa_engine()`, `lsa()`

Examples

```
my_engine <- function(transitions, ...) {
  get_lsa_engine("classical")$fn(transitions, ...)
}
register_lsa_engine("my_classical", my_engine, "Classical test alias")
fit <- lsa(engagement, engine = "my_classical")
unregister_lsa_engine("my_classical")
fit
```

reliability_lsa

Split-Half Reliability for an LSA Fit

Description

Estimates the reliability of an LSA network by repeated random split-half resampling of sequences. Each replicate draws two disjoint halves of the sequences without replacement, refits the engine on each half, and computes the correlation between the two half-network edge-weight vectors. Returns the distribution of replicate correlations plus a point summary.

Usage

```
reliability_lsa(fit, ...)

## S3 method for class 'lsa'
reliability_lsa(
  fit,
  R = 100L,
  weights = c("prob", "count", "adj_res"),
  method = c("pearson", "spearman"),
  parallel = FALSE,
  n_cores = NULL,
  verbose = FALSE,
  ...
)

## S3 method for class 'lsa_group'
reliability_lsa(fit, ...)
```

Arguments

<code>fit</code>	An lsa object returned by <code>lsa()</code> . Must be built from event-level data (sequences), not from a pre-computed transition matrix.
<code>...</code>	Reserved.
<code>R</code>	Integer. Number of split-half replicates. Default 100.
<code>weights</code>	Character. Which edge matrix to correlate across halves: "prob" (default), "count", or "adj_res".
<code>method</code>	Character. Correlation method: "pearson" (default) or "spearman".
<code>parallel</code>	Logical. Use multi-core resampling. Default FALSE.
<code>n_cores</code>	Integer. Worker count when <code>parallel = TRUE</code> .
<code>verbose</code>	Logical. Print progress every 100 replicates.

Value

An object of class `c("lsa_reliability", "list")` with:

correlations Numeric vector of length R: the split-half correlation of each replicate.

mean, sd Mean and standard deviation of the finite replicate correlations.

ci_low, ci_high Empirical 2.5% and 97.5% quantiles.

R, weights, method, n_sequences Recipe metadata.

fit Reference to the original fit.

References

Epskamp, S., Borsboom, D., & Fried, E. I. (2018). Estimating psychological networks and their accuracy: A tutorial paper. *Behavior Research Methods*, 50(1), 195-212.

For a grouped fit (`lsa_group`), reliability is estimated separately within each group and the per-group `lsa_reliability` objects are returned in an `lsa_reliability_group` container with its own print method.

See Also

[bootstrap_lsa\(\)](#), [stability_lsa\(\)](#), [permute_lsa\(\)](#)

Examples

```
fit <- lsa(engagement, engine = "classical")
rel <- reliability_lsa(fit, R = 50)
rel
```

stability_lsa

Case-Drop Stability for an LSA Fit

Description

Resamples the underlying sequence data **without** replacement at a specified retention proportion, refits the engine on each subsample, and records which edges remain significant at the recipe's alpha threshold. Returns a per-edge "stability" proportion: the fraction of subsamples in which the edge was significant. Edges with stability $\geq$ `min_stable` (default 0.95) are flagged as robust.

Usage

```
stability_lsa(
  fit,
  R = 500L,
  proportion = 0.8,
  min_stable = 0.95,
  parallel = FALSE,
  n_cores = NULL,
  verbose = FALSE,
  ...
)
```

Arguments

<code>fit</code>	An <code>lsa</code> object returned by lsa() .
<code>R</code>	Integer. Number of subsamples. Default 500.
<code>proportion</code>	Numeric in (0, 1). Fraction of cases retained per subsample. Default 0.8.
<code>min_stable</code>	Numeric in (0, 1). Stability threshold for the <code>stable</code> flag in the output edge frame. Default 0.95.
<code>parallel</code>	Logical. Use multi-core resampling. Default FALSE.

n_cores Integer. Worker count when parallel = TRUE.
verbose Logical. Print progress every 100 replicates.
... Reserved.

Value

An object of class `c("lsa_stability", "list")` with:

- edges** Tidy per-edge data frame with `observed_sig` (whether the cell was significant in the original fit), `stability` (fraction of subsamples in which the cell was significant), and `stable` (`stability >= min_stable`).
- stability_matrix** $R \times K^2$ 0/1 matrix recording per-cell significance across replicates.
- R, proportion, min_stable** Recipe metadata.
- fit** Reference to the original fit.

See Also

[bootstrap_lsa\(\)](#), [permute_lsa\(\)](#)

Examples

```
fit <- lsa(engagement, engine = "classical")
st <- stability_lsa(fit, R = 100)
head(as.data.frame(st))
```

tests	<i>Tablewise Independence Tests of an LSA Fit (Tidy)</i>
-------	--

Description

The global tests of independence (likelihood-ratio G^2 and Pearson chi-square) as a one-row-per-test `data.frame`.

Usage

```
tests(fit)

## S3 method for class 'lsa'
tests(fit)

## S3 method for class 'lsa_group'
tests(fit)
```

Arguments

fit An lsa fit from [lsa\(\)](#).

Value

A data.frame with columns test ("1rx2" / "x2"), statistic, df, p. Tests the engine did not compute are omitted.

See Also

`transitions()`, `nodes()`

Examples

```
tests(lsa(group_regulation))
```

transfer_entropy	<i>Directed transfer entropy for categorical sequences (experimental)</i>
------------------	---

Description

Experimental. Transfer entropy (Schreiber, 2000) measures *directed* predictive coupling: how much knowing the source's past reduces uncertainty about the target's future, **beyond what the target's own past already explains**. Because it conditions on the target's own history, transfer entropy is immune to the autocorrelation confound that inflates plain lagged association when a process has strong momentum.

Unlike `lsa()`'s Yule's Q / adjusted residuals, transfer entropy is **sign-blind**: a large value means "strong directed predictive structure", which may be *facilitating* or *suppressing*. Read it alongside the signed measures from `transitions()` to interpret direction of effect.

Two modes:

- **State-flow network** (default, `y = NULL`): given categorical sequences, returns directed transfer entropy between every ordered pair of states, via a binary occupancy decomposition. Note: with very few states the non-target source channels become redundant; prefer the bivariate mode or a larger alphabet when that matters.
- **Bivariate** (`y` supplied): full-alphabet transfer entropy between two aligned categorical series, in both directions.

Usage

```
transfer_entropy(
  x,
  y = NULL,
  lag = 1L,
  history = 1L,
  test = c("surrogate", "none"),
  R = 199L,
  normalize = TRUE,
  seed = NULL
)
```

Arguments

x	Categorical sequence data: a vector (one sequence), or a matrix / data.frame with one sequence per row and one time-step per column (NA-padded), exactly the shape <code>lsa()</code> consumes.
y	Optional second series, same shape as x, for bivariate transfer entropy. When NULL, the directed state-flow network of x is returned.
lag	Integer ≥ 1 . Prediction horizon: the target's future is taken lag steps ahead. Default 1.
history	Integer ≥ 1 . Order of the target's own history conditioned on (and combined into a composite symbol). Default 1.
test	"surrogate" (default) runs a source-permutation null to give a p-value and the bias-corrected <i>effective</i> transfer entropy; "none" skips it.
R	Integer. Number of surrogate permutations. Default 199.
normalize	Logical. Add <code>te_normalised</code> , transfer entropy as a share of the target's leftover uncertainty $H(\text{future} \mid \text{history})$, in $[0, 1]$. Default TRUE.
seed	Optional integer seed for the surrogate test.

Value

A tidy data.frame, one row per ordered pair, with columns `from`, `to`, `te` (bits), `te_effective` (surrogate-debiased), `te_normalised` (0-1, if `normalize`), `p` (surrogate p-value), and `n` (pooled transitions used). Rows are ordered by descending `te`.

References

Schreiber, T. (2000). Measuring information transfer. *Physical Review Letters*, 85(2), 461-464.

Examples

```
# Directed information-flow network over engagement states
transfer_entropy(engagement, test = "none")

# Bivariate transfer entropy between two aligned series
a <- c("calm", "calm", "tense", "tense", "calm", "tense", "tense", "calm")
b <- c("low", "low", "low", "high", "high", "low", "high", "high")
transfer_entropy(a, b, test = "none")
```

transition_probabilities

Transition-Probability Matrix of an LSA Fit

Description

The row-stochastic transition-probability matrix $P(\text{to} \mid \text{from})$: each row holds the distribution over next states out of a state, so the entries in a row sum to 1 (up to structural zeros). This is the transition-probability matrix a Transition Network Analysis reads; pair it with `initial()` for the initial-state probabilities.

Usage

```
transition_probabilities(fit)

## S3 method for class 'lsa'
transition_probabilities(fit)

## S3 method for class 'lsa_group'
transition_probabilities(fit)
```

Arguments

`fit` An lsa fit from `lsa()`.

Value

A square numeric matrix with from states in rows and to states in columns. For an `lsa_group`, a named list of such matrices, one per group.

See Also

`initial()` (initial-state probabilities), `transitions()` (the tidy per-edge view), `nodes()`

Examples

```
transition_probabilities(lsa(group_regulation))
```

transitions

Transitions of an LSA Fit or an Inference Result (Tidy)

Description

The canonical way to read the per-transition table of a fit or of any inference result as a tidy one-row-per-transition `data.frame`. `transitions(x)` returns every transition; the arguments narrow it.

Usage

```
transitions(fit, ...)

## S3 method for class 'lsa'
transitions(
  fit,
  significant = FALSE,
  direction = c("any", "over", "under"),
  min_count = NULL,
  alpha = NULL,
```

```

    sort = c("none", "strength", "count", "prob"),
    ...
)

## S3 method for class 'lsa_group'
transitions(
  fit,
  significant = FALSE,
  direction = c("any", "over", "under"),
  min_count = NULL,
  alpha = NULL,
  sort = c("none", "strength", "count", "prob"),
  ...
)

## S3 method for class 'lsa_bootstrap'
transitions(fit, significant = FALSE, sort = c("none", "strength"), ...)

## S3 method for class 'lsa_certainty'
transitions(fit, significant = FALSE, sort = c("none", "strength"), ...)

## S3 method for class 'lsa_stability'
transitions(fit, significant = FALSE, sort = c("none", "strength"), ...)

## S3 method for class 'lsa_permutation'
transitions(fit, significant = FALSE, sort = c("none", "strength"), ...)

## S3 method for class 'lsa_comparison'
transitions(fit, significant = FALSE, sort = c("none", "strength"), ...)

## S3 method for class 'lsa_comparison_pairwise'
transitions(fit, significant = FALSE, sort = c("none", "strength"), ...)

## S3 method for class 'lsa_lags'
transitions(
  fit,
  significant = FALSE,
  direction = c("any", "over", "under"),
  min_count = NULL,
  alpha = NULL,
  sort = c("none", "strength", "count", "prob"),
  ...
)

```

Arguments

fit	An lsa fit from lsa() , a grouped lsa_group, a multi-lag lsa_lags from lsa_lags() , or an inference result from bootstrap_lsa() , certainty_lsa() , stability_lsa() , permute_lsa() , compare_lsa() or bayes_compare_lsa() .
-----	--

...	Arguments passed to the method.
significant	Logical. Keep only the transitions the method flags as significant or stable. Default FALSE (keep all).
direction	One of "any" (default), "over" (over-represented: significant with a positive residual), or "under" (under-represented: significant with a negative residual). Selecting a direction implies significant = TRUE.
min_count	Optional integer. Keep only transitions observed at least this many times. Default NULL (no count filter).
alpha	Significance threshold. Default NULL, which uses the alpha recorded on the fit (fit\$params\$alpha).
sort	Row ordering. "none" (default) keeps the matrix (column-major) order; "strength" orders by the method's signed effect, "count" by observed count, "prob" by transition probability – each descending, so the table reads strongest-first.

Details

Applied to a fit from `lsa()` the table holds the estimated transition statistics. Applied to the result of `bootstrap_lsa()`, `certainty_lsa()`, `stability_lsa()`, `permute_lsa()`, `compare_lsa()` or `bayes_compare_lsa()` the table holds that method's per-transition quantities, and significant = TRUE keeps the transitions the method itself flags: resampling-stable for `bootstrap_lsa()` and `certainty_lsa()`, stable across replicates for `stability_lsa()`, and below the significance threshold for `permute_lsa()`, `compare_lsa()` and `bayes_compare_lsa()`.

Value

A data.frame, one row per transition. For a fit the columns are from, to (the source and target **state names**), lag, count, expected, prob (row-conditional), prob_col (column-conditional), adj_res, p, yules_q, kappa, kappa_z, kappa_p, lift, sign, significant. Engines that compute extra per-cell statistics append them as further columns (e.g. the two-cell engine adds odds_ratio, log_or, log_or_se). An inference result carries from, to and that method's own quantities. A grouped fit gains a leading group column. Row names are reset.

See Also

`lsa()`, `nodes()`, `tests()`

Examples

```
fit <- lsa(group_regulation)
transitions(fit)                # all transitions
transitions(fit, significant = TRUE) # significant ones
transitions(fit, direction = "over") # over-represented
transitions(fit, min_count = 500)  # frequently observed

transitions(certainty_lsa(fit))    # analytic CIs per edge
transitions(certainty_lsa(fit), significant = TRUE) # the certain ones
```

unregister_lsa_engine *Remove a Registered LSA Engine*

Description

Remove a Registered LSA Engine

Usage

```
unregister_lsa_engine(name)
```

Arguments

name	Character scalar. The engine's identifier.
------	--

Value

Invisibly NULL.

See Also

[register_lsa_engine\(\)](#)

Examples

```
my_engine <- function(transitions, ...) {  
  get_lsa_engine("classical")$fn(transitions, ...)  
}  
register_lsa_engine("temporary_engine", my_engine, "Temporary alias")  
unregister_lsa_engine("temporary_engine")
```

Index

* datasets

- ai_long, 3
- engagement, 14
- group_regulation, 15
- group_regulation_long, 16

ai_long, 3

as.data.frame(), 4, 5, 7, 10, 25

as.data.frame.lsa_comparison, 4

as.data.frame.lsa_comparison_pairwise
(as.data.frame.lsa_comparison),
4

as.data.frame.lsa_data, 4

as.data.frame.lsa_reliability, 5

as.data.frame.lsa_reliability_group
(as.data.frame.lsa_reliability),
5

bayes_compare_lsa, 6

bayes_compare_lsa(), 47, 48

bootstrap_lsa, 7

bootstrap_lsa(), 10, 11, 13, 30, 35, 36, 42,
43, 47, 48

certainty_lsa, 10

certainty_lsa(), 7, 32, 47, 48

cograph::plot_chord(), 30, 34, 35

cograph::splot(), 30, 39

compare_lsa, 11

compare_lsa(), 6, 7, 32, 33, 47, 48

engagement, 14

get_lsa_engine, 15

get_lsa_engine(), 19, 40

group_regulation, 15, 16, 17

group_regulation_long, 16

lag_profile, 18

list_lsa_engines, 18

list_lsa_engines(), 15, 22, 40

lsa(lsa_classical), 19

lsa(), 3, 7, 8, 10, 17, 18, 23, 25, 26, 28, 29,
31, 34, 37, 38, 40–48

lsa_bidirectional(lsa_classical), 19

lsa_classical, 19

lsa_data, 22, 27

lsa_data(), 5, 14, 20, 22, 27, 28

lsa_ipf, 24

lsa_lags, 25

lsa_lags(), 18, 20, 47

lsa_nonparallel_dominance
(lsa_classical), 19

lsa_parallel_dominance(lsa_classical),
19

lsa_to_tna, 26

lsa_transitions, 27

lsa_transitions(), 22, 23, 40

lsa_two_cell(lsa_classical), 19

nodes, 28

nodes(), 17, 21, 44, 46, 48

permute_lsa, 29

permute_lsa(), 9, 12, 13, 42, 43, 47, 48

plot(), 7, 21

plot.lsa, 30

plot.lsa(), 33, 35–37, 39

plot.lsa_bootstrap(plot_forest), 35

plot.lsa_certainty, 32

plot.lsa_comparison, 32

plot.lsa_comparison_pairwise
(plot.lsa_comparison), 32

plot.lsa_group(plot.lsa), 30

plot_chords, 34

plot_chords(), 30, 31, 37

plot_forest, 35

plot_forest(), 10, 11, 31, 32, 36, 37

plot_polar, 36
plot_polar(), 30, 31, 36
plot_transitions, 38
plot_transitions(), 21, 30, 31, 35

register_lsa_engine, 39
register_lsa_engine(), 15, 19, 20, 22, 49
reliability_lsa, 40
reliability_lsa(), 22

stability_lsa, 42
stability_lsa(), 9, 11, 30, 42, 47, 48
stats::p.adjust(), 7, 12
summary(), 21

tests, 43
tests(), 21, 28, 48
transfer_entropy, 44
transition_probabilities, 45
transition_probabilities(), 17, 22, 39
transitions, 46
transitions(), 17, 21, 22, 25, 28, 31, 35, 39,
44, 46

unregister_lsa_engine, 49
unregister_lsa_engine(), 40