Package 'lagdynamics'

November 18, 2026

Title Lag Sequential Analysis, Dynamics, and Lag Transition Networks

Version 0.35.0

Description Implements lag sequential analysis of categorical event and sequence data. For each ordered pair of states it tests whether one state follows another at a given lag more or less often than expected under independence, reporting adjusted residuals, transitional probabilities and Yule's Q in tidy data frames. Five test engines are provided (classical, two-cell, bidirectional, parallel dominance and non-parallel dominance). Transitions can be evaluated with bootstrap intervals, analytic certainty, split-half reliability, case-drop stability and permutation tests, and groups compared by permutation or by a Bayesian Dirichlet-multinomial contrast. Supports long-format event logs, sequence objects, multiple lags, structural zeros, transition and initial probabilities, plots of lag transition networks, and a directed transfer-entropy measure. The method follows Sackett and others (1979) <doi:10.3758/BF03205679>.

License MIT + file LICENSE

URL https://github.com/mohsaqr/lagdynamics, https://pak.dynasite.org/lagdynamics/

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

Language en-US

Encoding UTF-8

Depends R (>= 4.1.0)

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

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

Config/testthat/edition 3

LazyData true

VignetteBuilder knitr

Config/roxygen2/version 8.1.1

NeedsCompilation no

Author Mohammed Saqr [aut, cre, cph] (ORCID:
<https://orcid.org/0000-0001-5881-3109>),
Sonsoles López-Pernas [aut] (ORCID:
<https://orcid.org/0000-0002-9621-1392>)

Maintainer Mohammed Saqr <saqr@saqr.me>

Repository CRAN

Date/Publication 2026-11-18 10:10:02 UTC

Contents

ai_long . . . 3
as.data.frame.lsa_comparison . . . 4
as.data.frame.lsa_data . . . 5
as.data.frame.lsa_reliability . . . 6
bayes_compare_lsa . . . 8
bootstrap_lsa . . . 10
certainty_lsa . . . 13
compare_lsa . . . 15
engagement . . . 18
get_lsa_engine . . . 19
group_regulation . . . 20
group_regulation_long . . . 20
initial . . . 22
lag_profile . . . 23
list_lsa_engines . . . 24
lsa . . . 24
lsa-inference-methods . . . 28
lsa-methods . . . 29
lsa_data . . . 32
lsa_ipf . . . 33
lsa_lags . . . 35
lsa_to_tna . . . 36
lsa_transitions . . . 37
nodes . . . 38
permute_lsa . . . 39
plot.lsa . . . 41
plot.lsa_certainty . . . 43
plot.lsa_comparison . . . 44
plot_chords . . . 45
plot_forest . . . 47
plot_polar . . . 48
plot_transitions . . . 50
register_lsa_engine . . . 51
reliability_lsa . . . 53
stability_lsa . . . 55
tests . . . 57
transfer_entropy . . . 58

transitions 60
transition_probabilities 62
unregister_lsa_engine 63

Index **65**

`ai_long` *Coded AI behaviours in human-AI programming sessions*

Description

`ai_long` is a coded dataset, in long format, of the behaviours of an AI coding assistant in "vibe-coding" sessions, in which a person builds software by instructing the assistant in natural language. Each row is one coded AI behaviour, with the project and session it belongs to and its position in the session. A single assistant message can carry several codes, which are ordered within the message.

The log contains 8,551 coded events in 428 sessions from 34 projects. Eight behaviour codes are grouped into three clusters: `Action` (`Delegate`, `Execute`, `Investigate`), `Communication` (`Ask`, `Explain`, `Report`) and `Repair` (`Plan`, `Repair`). With `session = "session_id"`, `action = "code"` and `order = "order_in_session"`, `lsa()` reads each session as one sequence and tests which behaviour tends to follow which.

Usage

```
ai_long
```

Format

A data frame with 8,551 rows (coded events) and 9 columns:

`message_id` Integer. Identifier of the assistant message.

`project` Character. Project identifier (34 projects).

`session_id` Character. Session identifier (428 sessions, each within one project).

`timestamp` Integer. Unix time in seconds.

`session_date` Character. Session date as `YYYY-MM-DD`.

`code` Character. AI behaviour code (8 values).

`cluster` Character. Behaviour cluster (3 values).

`code_order` Integer. Position of the code within its message.

`order_in_session` Integer. Position of the event within its session.

Source

Taken without modification from `Nestimate::ai_long` (**Nestimate** 0.7.7, MIT license).

See Also

`lsa()` and its long-format arguments; group_regulation_long for another event log.

Examples

```
fit <- lsa(ai_long, session = "session_id", action = "code",
           order = "order_in_session")
transitions(fit, direction = "over", sort = "strength")
```

`as.data.frame.lsa_comparison` *Read the tables of a group comparison*

Description

`as.data.frame()` returns the tables of a group comparison from `compare_lsa()` or `bayes_compare_lsa()` as a plain data frame, and `print()` shows a short summary of the comparison. A comparison holds two tables, chosen with `what`: the comparison of every transition and the omnibus result for the structure as a whole. The per-transition table is the one `transitions()` reads; use `transitions()` when you want to keep only the significant transitions or order them by the size of the difference.

Usage

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

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

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

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

Arguments

x	A comparison from compare_lsa() or bayes_compare_lsa() (class lsa_comparison or lsa_comparison_pairwise).
row.names, optional	Not used; present for consistency with the generic.
what	"edges" (default): one row per transition. "global": the omnibus result, one row per pair of groups.
...	Not used.

Value

as.data.frame() returns a data frame.

- what = "edges": one row per transition (and per pair of groups when more than two groups were compared, with leading group_a, group_b columns); the columns are listed under *Value* in compare_lsa() and bayes_compare_lsa().
- what = "global": one row per pair of groups with group_a, group_b and, for compare_lsa(), the omnibus statistic (sum of squared differences), its permutation p_value and either the number of permutations R (two groups) or the across-pairs adjusted p_adj (more than two groups); for bayes_compare_lsa(), n_credible, the number of credibly different transitions.

print() returns x invisibly.

See Also

compare_lsa(), bayes_compare_lsa(), transitions().

Examples

```
g <- lsa(group_regulation_long, actor = "Actor", action = "Action",
         time = "Time", group = "Achiever")
cmp <- compare_lsa(g, R = 50, adjust = "BH")
cmp
as.data.frame(cmp, what = "global")
head(as.data.frame(cmp))
```

---

as.data.frame.lsa_data

Read the events of canonical sequence data as a table

Description

as.data.frame() returns the canonical sequence data from lsa_data(), every event with its sequence and position, as a tidy table, which shows how the input was read: which rows became sequences, where missing values were dropped, and which labels were assigned. For input given as a transition-count matrix, it returns one row per cell instead. print() shows the input source, the number of states, sequences, events and lag-1 transitions.

Usage

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

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

Arguments

`x`	An `lsa_data` object from `lsa_data()`.
`row.names`, `optional`, `...`	Ignored; present for consistency with the generic.

Value

`as.data.frame()`: a `data.frame` with one row per event and the columns `seq_id`, `index` (position within the sequence) and `state`; or, for transition-matrix input, one row per cell and the columns `from`, `to` and `count`. `print()` returns `x` invisibly.

See Also

`lsa_data()`, `lsa_transitions()`

Examples

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

---

as.data.frame.lsa_reliability

Read the tables of a split-half reliability result

Description

`as.data.frame()` returns the tables of a `reliability_lsa()` result, which holds the split-half correlation of every random split: these correlations, one row per split, or a one-row summary of their distribution, chosen with `what`. For a grouped result the tables of the groups are stacked.

Usage

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

```
  what = c("replicates", "summary"),
  ...
)

## 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 from reliability_lsa().
row.names, optional	Not used; present for consistency with the generic.
what	"replicates" (default): one row per split. "summary": one row summarizing all splits.
...	For a grouped result, passed on to each group (use it for what); otherwise not used.

Value

A data frame.

- what = "replicates": one row per split, with replicate and correlation; a failed replicate (a half with no transitions, or no variation to correlate) is kept as NA.
- what = "summary": one row with weights, method, R, n_failed, mean, sd, ci_low and ci_high (2.5% and 97.5% quantiles of the finite correlations).

A grouped result gains a leading group column.

See Also

reliability_lsa().

Examples

```
rel <- reliability_lsa(lsa(engagement), R = 20)
as.data.frame(rel) |> head()
as.data.frame(rel, what = "summary")

g <- lsa(group_regulation_long, actor = "Actor", action = "Action",
         time = "Time", group = "Achiever")
grel <- reliability_lsa(g, R = 10)
as.data.frame(grel, what = "summary")
```

`bayes_compare_lsa` *Compare transition probabilities between groups with a Bayesian model*

Description

`bayes_compare_lsa()` compares the transition probabilities of groups with a Bayesian model: given the counts, it estimates the plausible range of the true difference between two groups' transition probabilities and how precisely that difference is estimated. Under a Dirichlet-Multinomial model the outgoing transitions of each state are a multinomial draw whose probabilities have a Dirichlet posterior, so each transition probability has a Beta posterior in each group. A transition whose source state is rarely visited gets a wide interval, even when its estimated probabilities look far apart. A permutation test (`compare_lsa()`) instead tests whether a group difference is larger than chance.

For every transition, `bayes_compare_lsa()` computes the posterior mean difference of the transition probability between two groups (exact), its credible interval and the probability of direction (the posterior probability of the more probable sign of the difference), both from Monte Carlo draws of the Beta posteriors, and a standardised effect size. A transition is credibly different when its credible interval excludes zero, the absolute mean difference exceeds `mean_threshold`, and the interval end nearest zero lies further from zero than `bound_threshold`; the thresholds keep detectable but negligible differences out. With more than two groups every pair is compared. No resampling of sequences is involved, so the comparison is fast.

The model treats transitions as independent draws and compares transition probabilities only. For a test that respects the dependence between events of one sequence, or that compares adjusted residuals or odds ratios, use `compare_lsa()`. For the posterior of one group's transition probabilities, use `certainty_lsa()`.

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
)

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

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

Arguments

x	A grouped fit from `lsa(data, group = )` with two or more groups, or the `lsa` fit of the first group. For the `print()` methods, the comparison result.
y	The `lsa` fit of the second group when `x` is a single fit (the groups are then labelled `"a"` and `"b"`). Default `NULL`.
prior	Dirichlet concentration added to every cell. Default `0.5` (the Jeffreys prior); `1` gives a uniform prior.
draws	Number of Monte Carlo draws for the credible intervals. Default `10000`.
ci	Credible-interval level. Default `0.95`.
mean_threshold	Minimum absolute posterior mean difference for a transition to be credibly different. Default `0.01`.
bound_threshold	Minimum distance from zero of the interval end nearest zero. Default `0.001`.
adjust	Multiple-comparison correction applied to the two-sided Bayesian p-value `2 * (1 - pd)` across transitions (and across all pairs for more than two groups); any method of `stats::p.adjust()`. Default `"none"`. It changes the `p_adj` column only: the `significant` flag is decided by the credible interval and the thresholds.
seed	Optional seed for the Monte Carlo draws. Default `NULL`. The session's random number state is restored on exit.
...	Not used.

Value

With two groups, an object of class `c("lsa_bayes", "lsa_comparison", "list")`; with more than two, an object of class `c("lsa_bayes_pairwise", "lsa_comparison_pairwise", "list")`. Read the per-transition table with `transitions()`: one row per transition (`from -> to` pair), and for more than two groups one row per transition and pair of groups, with the columns

- `group_a`, `group_b` (more than two groups only): the pair compared;
- `from`, `to`: source and target state;
- `prob_a`, `prob_b`: posterior mean transition probability in each group;
- `diff`: `prob_a - prob_b`; `ci_low`, `ci_high`: its credible interval;
- `pd`: probability of direction; `effect_size`: `diff` divided by its posterior standard deviation;
- `p_value`: `2 * (1 - pd)` (`NA` for transitions that cannot be estimated); `p_adj`: after `adjust`;
- `significant`: credibly different.

`transitions(x, significant = TRUE)` keeps the credibly different transitions; `sort = "strength"` orders by the absolute difference. `as.data.frame(x, what = "global")` gives the number of credibly different transitions per pair of groups (see `as.data.frame.lsa_comparison()`); `plot()` draws the differences. `print()` shows the settings and the number of credibly different transitions and returns `x` invisibly.

Conditions

Errors carry the class `lagdynamics_error` and one of:

- `lagdynamics_needs_events`: a fit was built from a transition matrix.
- `lagdynamics_incompatible_fits`: the groups differ in their states, engine, lag or structural zeros.
- `lagdynamics_bad_input`: `adjust` is not a method of `stats::p.adjust()`; `x` is neither a grouped fit nor an `lsa` fit with a second fit in `y`; `y` is given together with a grouped fit; or the grouped fit has fewer than two groups.

References

Johnston, L., & Jendoubi, T. (2026). How delivery mode reshapes resource engagement: A Bayesian differential network analysis. TNA Workshop 2026.

See Also

`transitions()` and `as.data.frame.lsa_comparison()` to read the result; `plot.lsa_comparison()` to draw it; `compare_lsa()` for the permutation test; `certainty_lsa()` for one group's posterior.

Examples

```
g <- lsa(group_regulation_long, actor = "Actor", action = "Action",
         time = "Time", group = "Achiever")
bc <- bayes_compare_lsa(g, adjust = "BH", seed = 1)
bc
transitions(bc, significant = TRUE, sort = "strength") |> head()

# Three courses: every pair
courses <- lsa(group_regulation_long, actor = "Actor", action = "Action",
               time = "Time", group = "Course")
bayes_compare_lsa(courses, draws = 2000, seed = 1) |>
  as.data.frame(what = "global")
```

---

bootstrap_lsa *Estimate bootstrap confidence intervals for transitions*

Description

`bootstrap_lsa()` estimates bootstrap confidence intervals for the transitions of a fit from `lsa()`: it resamples the sequences behind the fit and refits it with the same engine, lag, alternative, significance level and structural zeros. The non-parametric bootstrap (Efron, 1979) estimates how much the count, transition probability, adjusted residual and Yule's Q of each transition, all estimated from one sample of sequences, would vary over repeated samples from the same population: it draws many resamples of the data, refits the model on each, and uses the spread of the refitted values as the sampling distribution. A transition whose residual interval excludes zero is over- or under-represented in a way that survives resampling, not only in the sample at hand.

With two or more sequences, whole sequences are drawn with replacement, which keeps the dependence between events of one sequence intact. A single sequence is resampled with the stationary block bootstrap (Politis & Romano, 1994), which draws blocks of consecutive events of random length. For every transition it returns the bootstrap mean, standard error and percentile interval of the count, the transition probability, the adjusted residual and Yule's Q, a two-sided bootstrap p-value for the residual, and a flag for transitions whose residual interval excludes zero.

For an interval on transition probabilities without resampling, use `certainty_lsa()`; it is exact under a Dirichlet-Multinomial model but ignores dependence between the transitions of one sequence. For a p-value against the null hypothesis of no sequential structure, use `permute_lsa()`; for the share of subsamples in which a transition stays significant, use `stability_lsa()`.

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

`fit`	An `lsa` fit from `lsa()`, built from sequences (not from a transition matrix).
`R`	Number of bootstrap resamples. Default `1000`.
`level`	Resampling unit: `"auto"` (default; `"sequence"` when the fit has two or more sequences, `"event"` otherwise), `"sequence"` (whole sequences with replacement) or `"event"` (stationary block bootstrap of the event stream). A single sequence always uses `"event"`, with a warning.
`block_length`	Mean block length for `level = "event"`. Default `NULL`, the square root of the number of events, rounded up.
`level_alpha`	Confidence level of the percentile intervals. Default `0.95`.
`indices`	Optional matrix of resampling indices to replay, one row per resample: `R` x number of sequences (sequence indices) for `level = "sequence"`, or `R` x number of events (event positions) for `level = "event"`. Default `NULL` draws them at random.
`parallel`	Run the resamples on several cores. Default `FALSE`.
`n_cores`	Number of cores when `parallel = TRUE`. Default `NULL`, all cores but one.
`verbose`	Report progress every 100 resamples. Default `FALSE`.
`...`	Not used.

Value

An object of class `c("lsa_bootstrap", "list")`. Read it with `transitions()`, which returns a data frame with one row per transition (`from -> to` pair) and the columns:

- `from`, `to`: source and target state.
- `observed`: the observed count; `count_mean`, `count_se`, `count_ci_low`, `count_ci_high`: its bootstrap mean, standard error and interval.
- `adj_res_observed`, `adj_res_mean`, `adj_res_se`, `adj_res_ci_low`, `adj_res_ci_high`: the same for the adjusted residual; `adj_res_p_boot`: two-sided bootstrap p-value of the residual; `adj_res_stable`: `TRUE` when the residual interval excludes zero.
- `prob_observed`, `prob_mean`, `prob_ci_low`, `prob_ci_high`: the transition probability.
- `yules_q_observed`, `yules_q_mean`, `yules_q_ci_low`, `yules_q_ci_high`: Yule's Q.

`transitions(x, significant = TRUE)` keeps the transitions with `adj_res_stable = TRUE`; `sort = "strength"` orders by the absolute observed residual. `print()` shows the settings and the number of stable transitions (see lsa-inference-methods); `plot_forest()` draws the intervals. The object also stores the per-resample matrices, the settings, the resampling indices used (which can be passed back as `indices`) and the original fit.

Conditions

Errors carry the class `lagdynamics_error` and one of:

- `lagdynamics_needs_events`: the fit was built from a transition matrix, so there are no sequences to resample.
- `lagdynamics_bad_input`: `indices` is not a matrix of whole numbers with at least `R` rows, the expected number of columns and values in range, or `block_length` is not a single number of at least 1.

References

Efron, B. (1979). Bootstrap methods: Another look at the jackknife. *The 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

`transitions()` to read the result; `plot_forest()` to draw the intervals; `certainty_lsa()` for analytic intervals; `permute_lsa()`, `stability_lsa()` and `reliability_lsa()` for other checks of a fit; `compare_lsa()` to compare groups.

Examples

```
fit <- lsa(engagement)
bs <- bootstrap_lsa(fit, R = 100)
bs
transitions(bs, significant = TRUE, sort = "strength")
```

```
# A full analysis uses more resamples
bootstrap_lsa(fit, R = 1000) |>
  transitions(sort = "strength") |>
  head()
```

`certainty_lsa` *Compute the analytic certainty of transition probabilities*

Description

`certainty_lsa()` computes the analytic certainty of every transition probability of a fit from `lsa()`: how precisely the data pin down each transition probability, and which transitions are estimated precisely enough to be called certain. Under a Dirichlet-Multinomial model the outgoing transitions of each state are a multinomial draw whose probabilities have a Dirichlet posterior, so each transition probability has a Beta posterior with a closed-form mean, standard deviation and credible interval. A transition probability estimated from counts is uncertain in proportion to how often its source state was visited: a probability of 0.5 from 4 departures is far less certain than the same value from 400.

The posterior is computed with a Jeffreys prior by default and needs no resampling. Each transition gets its posterior mean, standard deviation and credible interval, and a posterior tail probability that decides whether it is certain. With `inference = "stability"` a transition is certain when its posterior stays within a band around the observed probability (by default 75% to 125% of it); with `inference = "threshold"` it is certain when its posterior lies above a minimum probability. A grouped fit gives one result per group.

The model treats the transitions as independent draws. When sequences differ strongly from one another (for example, a mixture of latent groups with long sequences) it reports more certainty than the data support; use `bootstrap_lsa()`, which resamples whole sequences, in that case. To compare two groups' transition probabilities with the same model, use `bayes_compare_lsa()`.

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
)

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

## S3 method for class 'lsa_certainty_group'
```

```
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

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

Arguments

fit	An lsa fit from lsa(), or a grouped fit from lsa(..., group =).
prior	Dirichlet concentration added to every cell. Default 0.5 (the Jeffreys prior).
level_alpha	Credible-interval level. Default 0.95. The certainty decision uses 1 - level_alpha as its threshold.
inference	"stability" (default): a transition is certain when little posterior mass lies outside consistency_range times its observed probability. "threshold": certain when little posterior mass lies below edge_threshold.
consistency_range	Two multipliers of the observed probability that bound the band for inference = "stability". Default c(0.75, 1.25).
edge_threshold	Minimum probability for inference = "threshold". Default NULL, the 10% quantile of the non-zero transition probabilities.
x	An lsa_certainty or lsa_certainty_group object, for the print() and as.data.frame() methods.
...	Not used.
row.names, optional	Not used; present for consistency with the generic.

Value

An object of class c("lsa_certainty", "lsa_bootstrap", "list"). Read it with transitions(), which returns a data frame with one row per transition (from -> to pair) and the columns from, to, observed (count), prob_observed, prob_mean (posterior mean), prob_se (posterior standard deviation), prob_ci_low, prob_ci_high (credible interval), p_value (posterior mass outside the band, or below the threshold), stable (certain), and adj_res_observed, adj_res_stable (the fit's adjusted residual and a copy of stable, used by plot_forest()). Transitions that cannot be estimated (a source state never left, or a structural zero) are NA and never certain. transitions(x, significant = TRUE) keeps the certain transitions.

For a grouped fit, an object of class lsa_certainty_group, a list with one lsa_certainty per group; as.data.frame() stacks their tables with a leading group column. print() shows the settings and the number of certain transitions and returns x invisibly.

References

Johnston, L., & Jendoubi, T. (2026). How delivery mode reshapes resource engagement: A Bayesian differential network analysis. TNA Workshop 2026.

See Also

transitions() to read the result; plot.lsa_certainty() and plot_forest() to draw it; bootstrap_lsa() for resampling intervals; bayes_compare_lsa() to compare groups under the same model.

Examples

```
fit <- lsa(engagement)
cert <- certainty_lsa(fit)
cert
transitions(cert, significant = TRUE)

# One table per group
g <- lsa(group_regulation_long, actor = "Actor", action = "Action",
         time = "Time", group = "Achiever")
gcert <- certainty_lsa(g)
gcert
head(as.data.frame(gcert))
```

`compare_lsa` *Compare transition structures between groups with a permutation test*

Description

`compare_lsa()` tests whether groups of sequences (high and low achievers, courses, conditions) differ in their transition structure, transition by transition and as a whole, with a permutation test. Groups can differ in which transitions they over- or under-produce. The permutation test checks whether an observed difference between groups is larger than would arise if group membership were unrelated to the sequences: the group labels of whole sequences are shuffled many times, the models are refitted, and the observed difference is set against the differences produced by the shuffles. Because whole sequences are permuted, the dependence between events of one sequence is kept and only the link between group and transition structure is tested.

For each transition, `compare_lsa()` compares a per-group measure, by default the log odds ratio of the transition against all other transitions, which does not grow with the number of transitions observed, so groups of different sizes can be compared. Each transition gets the group difference, a permutation p-value (Phipson & Smyth, 2010) and an adjusted p-value; transitions too rare to be compared are left untested. An omnibus test of overall difference, the sum of squared differences over the tested transitions, is computed from the same permutations, in the spirit of the network comparison test of van Borkulo et al. (2023). With more than two groups every pair is compared and the `adjust` correction is applied once across the p-values of all pairs.

The groups must share their states, engine, lag and structural zeros; fitting them together with `lsa(data, group = )` ensures this. For a Bayesian comparison of transition probabilities that reports the plausible size of each difference without resampling, use `bayes_compare_lsa()`. A difference between groups can also arise from a difference in their composition; check subgroups when such a confound is plausible.

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	A grouped fit from `lsa(data, group = )` with two or more groups, or the `lsa` fit of the first group.
y	The `lsa` fit of the second group when `x` is a single fit (the groups are then labelled `"a"` and `"b"`). Default `NULL`.
R	Number of label permutations per comparison. Default `1000`.
measure	Per-transition measure compared between groups: `"log_or"` (default; log odds ratio of the transition, with 0.5 added to every cell of its 2 x 2 table when one cell is empty), `"yules_q"` (also independent of sample size, but bounded at -1 and 1), `"adj_res"` (adjusted residual; grows with the number of transitions, so a message is shown when one group has more than 1.5 times the transitions of another), `"prob"` (transition probability), `"count"` or `"lift"` (observed / expected count).
adjust	Multiple-comparison correction, any method of `stats::p.adjust()` such as `"BH"`, `"holm"` or `"bonferroni"`. Default `"none"`.
min_count	Minimum combined count of a transition in the two groups for it to be tested. Default `5`; `0` tests every transition.
parallel	Run the permutations on several cores. Default `FALSE`.
n_cores	Number of cores when `parallel = TRUE`. Default `NULL`, all cores but one.
verbose	Report progress. Default `FALSE`.
...	Not used.

Value

With two groups, an object of class `c("lsa_comparison", "list")`; with more than two, an object of class `c("lsa_comparison_pairwise", "list")`. Read the per-transition table with `transitions()`: one row per transition (`from -> to` pair), and for more than two groups one row per transition and pair of groups, with the columns

- `group_a`, `group_b` (more than two groups only): the pair compared;
- `from`, `to`: source and target state;
- `<measure>_a`, `<measure>_b` (for example `log_or_a`): the measure in each group;
- `diff`: the difference, group a minus group b;
- `p_perm`: permutation p-value (`NA` when not tested);
- `p_adj`: p-value after `adjust`;

- `significant`: `p_adj` below the `alpha` of the fits.

`transitions(x, significant = TRUE)` keeps the significant transitions; `sort = "strength"` orders by the absolute difference. The omnibus test is read with `as.data.frame(x, what = "global")` (one row per pair of groups). `print()` shows the groups, the number of significant transitions and the omnibus test (see `as.data.frame.lsa_comparison()`); `plot()` draws the differences. The object also stores the permuted differences, the settings and the group fits.

Conditions

Errors carry the class `lagdynamics_error` and one of:

- `lagdynamics_needs_events`: a fit was built from a transition matrix, so there are no sequences to permute.
- `lagdynamics_incompatible_fits`: the groups differ in their states, engine, lag or structural zeros.
- `lagdynamics_bad_input`: `adjust` is not a method of `stats::p.adjust()`; `x` is neither a grouped fit nor an `lsa` fit with a second fit in `y`; `y` is given together with a grouped fit; or the grouped fit has fewer than two groups.

References

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.

van Borkulo, C. D., van Bork, R., Boschloo, L., Kossakowski, J. J., Tio, P., Schoevers, R. A., Borsboom, D., & Waldorp, L. J. (2023). Comparing network structures on three aspects: A permutation test. *Psychological Methods, 28*(6), 1273-1285.

See Also

`transitions()` and `as.data.frame.lsa_comparison()` to read the result; `plot.lsa_comparison()` to draw it; `bayes_compare_lsa()` for a Bayesian comparison; `permute_lsa()` to test one fit against no sequential structure; `lsa()` with `group =` to fit the groups.

Examples

```
g <- lsa(group_regulation_long, actor = "Actor", action = "Action",
         time = "Time", group = "Achiever")
cmp <- compare_lsa(g, R = 100, adjust = "BH")
cmp
transitions(cmp, significant = TRUE, sort = "strength") |> head()
as.data.frame(cmp, what = "global")

# A full analysis uses more permutations
compare_lsa(g, R = 1000, adjust = "BH")

# Three courses: every pair, corrected across all pairs
courses <- lsa(group_regulation_long, actor = "Actor", action = "Action",
               time = "Time", group = "Course")
```

```
compare_lsa(courses, R = 100, adjust = "BH") |>
  as.data.frame(what = "global")
```

`engagement` *Engagement states of students over time*

Description

`engagement` is a coded dataset, in wide format, of engagement: 138 students, each observed at 15 ordered time points and coded at each as `Active`, `Average` or `Disengaged`. Missing time points are `NA`. Two students have no observation at all, so a fit reports 136 sequences; the remaining sequences have on average about 14 observed states.

With three states and many short sequences, the data show the typical pattern of a lag sequential analysis: each state tends to persist, and moves between `Active` and `Disengaged` are rare. They are the main example of the help pages.

Usage

```
engagement
```

Format

A data frame with 138 rows (students) and 15 character columns named `1` to `15` (time points, in order). Values are `"Active"`, `"Average"`, `"Disengaged"` or `NA`.

Source

Derived from the synthetic `trajectories` example data of the **Nestimate** package (https://github.com/mohsaqr/Nestimate, MIT license), with the states kept and the matrix stored as a data frame.

See Also

group_regulation for a larger set of sequences; `lsa()`.

Examples

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

`get_lsa_engine` *Retrieve a registered engine*

Description

`get_lsa_engine()` returns the registry entry of one engine: its name, its function, its description and the packages it requires. It is the lookup `lsa()` performs before fitting, and it is useful for inspecting a built-in engine's function or for checking that a registered engine can run in the current session. To see which engines exist, use `list_lsa_engines()`.

Usage

```
get_lsa_engine(name)
```

Arguments

`name`	Name of the engine.

Value

A list with the elements `name`, `fn`, `description` and `requires`, as given to `register_lsa_engine()`.

Conditions

Errors carry the class `lagdynamics_error` and one of:

- `lagdynamics_unknown_engine`: no engine is registered under `name`.
- `lagdynamics_missing_package`: a package the engine requires is not installed.

See Also

`list_lsa_engines()`, `register_lsa_engine()`

Examples

```
str(get_lsa_engine("classical"), give.attr = FALSE)
```

group_regulation *Regulation actions in collaborative learning, as sequences*

Description

group_regulation is a coded dataset, in wide format, of regulation during collaborative learning: each row is one sequence of coded actions and each column an ordered position. The nine actions are adapt, cohesion, consensus, coregulate, discuss, emotion, monitor, plan and synthesis. Sequences are 2 to 26 actions long; shorter sequences are padded with NA.

With nine states and 2,000 sequences, the data give a dense transition table in which most transitions depart from independence, and they are used to show lag transition networks and group comparisons. The same actions are available as an event log in group_regulation_long.

Usage

```
group_regulation
```

Format

A data frame with 2,000 rows (sequences) and 26 character columns T1 to T26 (positions, in order).

Source

Identical to tna::group_regulation (**tna** package, MIT license).

See Also

group_regulation_long for the event log; engagement; lsa().

Examples

```
fit <- lsa(group_regulation)
transitions(fit, direction = "over", sort = "strength") |> head()
```

group_regulation_long *Regulation actions in collaborative learning, as an event log*

Description

group_regulation_long is a coded dataset, in long format, of regulation during collaborative learning: each row is one coded action of one student, with the student's collaboration group, course and achievement level and the time of the action. It contains the nine actions of group_regulation (adapt, cohesion, consensus, coregulate, discuss, emotion, monitor, plan, synthesis) and serves to show how lsa() reads an event log and fits and compares groups.

The log holds 27,533 actions of 2,000 students in 200 collaboration groups of 10 students each. Achievement (High or Low, 1,000 students each) is constant within a student, and the course (A, B or C) is constant within a group, so either can be passed to the group argument of lsa() for a between-group comparison.

Usage

```
group_regulation_long
```

Format

A data frame with 27,533 rows (actions) and 6 columns:

Actor Integer. Student identifier (2,000 students).

Achiever Character. Achievement level, High or Low.

Group Numeric. Collaboration group (200 groups).

Course Character. Course, A, B or C.

Time POSIXct. Time of the action.

Action Character. Regulation action (9 values).

Source

Taken without modification from tna::group_regulation_long (**tna** package, MIT license).

See Also

group_regulation for the wide sequence table of the same actions; lsa(); compare_lsa().

Examples

```
fit <- lsa(group_regulation_long, actor = "Actor", action = "Action",
           time = "Time")
transitions(fit, direction = "over", sort = "strength") |> head()

# One model per achievement level
gfit <- lsa(group_regulation_long, actor = "Actor", action = "Action",
            time = "Time", group = "Achiever")
summary(gfit)
```

`initial` *Read the initial-state distribution of a fit*

Description

`initial()` returns the initial-state distribution of a fit, one row per state: the proportion of sequences that begin in each state. It describes how episodes open, which the transition table cannot show, and it completes the description of the process as a first-order Markov chain together with `transition_probabilities()`.

A fit built from a transition-count matrix has no sequences and so no initial states; the table then has no rows.

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()`, or a grouped fit from `lsa(..., group = )`.

Value

A `data.frame` with one row per state and the columns `state` and `init_prob` (summing to 1). A grouped fit gains a leading `group` column.

See Also

`transition_probabilities()`, `nodes()`, `transitions()`

Examples

```
initial(lsa(engagement))
```

`lag_profile` *Trace one transition across lags*

Description

`lag_profile()` traces one `from -> to` transition across lags, one row per lag with the count, the transition probability, the adjusted residual and its p-value. This lag profile shows how the association between two states changes with the distance between them: whether a transition that is over-represented at lag 1 persists, decays or reverses at lags 2, 3 and beyond (Sackett, 1979). For example, it shows whether disengagement is still unlikely several time points after an active one, or only at the next.

`lag_profile()` fits the lags itself from raw data, or reads an existing `lsa_lags()` result.

Usage

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

Arguments

`x`	Sequences in any form accepted by `lsa()`, or an `lsa_lags()` result.
`from`, `to`	State names of the transition.
`lags`	Integer vector of lags (default `1:3`); ignored when `x` is an `lsa_lags` result.
`...`	Passed to `lsa_lags()` when `x` is raw data.

Value

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

Conditions

Errors carry the class `lagdynamics_error` and `lagdynamics_bad_input` when `from` or `to` is not a state of the data.

References

Sackett, G. P. (1979). The lag sequential analysis of contingency and cyclicity in behavioral interaction research. In J. D. Osofsky (Ed.), *Handbook of infant development* (pp. 623-649). Wiley.

See Also

`lsa_lags()`, `transitions()`

Examples

```
lag_profile(engagement, from = "Active", to = "Disengaged", lags = 1:4)
```

list_lsa_engines *List the registered engines*

Description

`list_lsa_engines()` lists the engines available in the session, the five built-in ones and any added with `register_lsa_engine()`, with a one-line description of each. Each engine of `lsa()` implements one test of the transition table. Use the list to choose the `engine` argument of `lsa()`.

Usage

```
list_lsa_engines()
```

Value

A `data.frame` with one row per registered engine, in alphabetical order, and the columns `name`, `description` and `requires` (comma-separated package names, `""` when none).

See Also

`lsa()`, `register_lsa_engine()`, `get_lsa_engine()`

Examples

```
list_lsa_engines()
```

lsa *Fit a lag sequential analysis model*

Description

`lsa()` fits a lag sequential analysis model to event sequences, wide sequence tables, transition-count matrices, long event logs, or sequence objects from other packages. Lag sequential analysis (Sackett, 1979; Bakeman & Gottman, 1997) tests whether, in a stream of coded events, one state follows another more or less often than chance predicts.

The transitions observed at a given lag are counted into a table of antecedent (`from`) by consequent (`to`) states; an expected table is derived under the hypothesis that the two are independent; and each cell is tested with an adjusted residual (Haberman, 1979), a standard normal deviate that is positive when the transition is over-represented and negative when it is avoided. The significant cells define a lag transition network, and the tablewise likelihood-ratio and Pearson tests show whether there is any sequential dependence at all. For every cell, `lsa()` estimates the observed and expected counts, the transition probability, the adjusted residual and its p-value, Yule's Q, kappa and lift. The `engine` selects the test: the classical analysis of Bakeman and Quera (1995) by default; a two-cell (2 x 2) test with odds ratios; Sackett's bidirectional test on the symmetrised table; or the parallel

and non-parallel dominance tests of Sackett (1979) and Wampold (1984). Self-transitions or any other cells that cannot occur by construction are declared as structural zeros, and the expected table is then fitted by iterative proportional fitting (Christensen, 1997). With `group`, one model is fitted per group on a shared state set so the groups can be compared.

It returns an `lsa` object, or with `group` an `lsa_group` object holding one fit per group.

The fit is read with `transitions()`, `nodes()`, `tests()`, `initial()` and `transition_probabilities()`, and drawn with `plot.lsa()`. To fit several lags at once use `lsa_lags()`; to assess the transitions use `bootstrap_lsa()`, `certainty_lsa()` or `permute_lsa()`; to compare groups use `compare_lsa()`. The engine wrappers `lsa_classical()`, `lsa_two_cell()`, `lsa_bidirectional()`, `lsa_parallel_dominance()` and `lsa_nonparallel_dominance()` are `lsa()` with `engine` fixed and return the same object.

Usage

```
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(),
  ...
)

lsa_classical(data, ...)

lsa_two_cell(data, ...)

lsa_bidirectional(data, ...)

lsa_parallel_dominance(data, ...)

lsa_nonparallel_dominance(data, ...)
```

Arguments

data	Sequences in any form accepted by lsa_data(): a vector, a list of sequences, a wide table (one row per sequence), a square transition-count matrix, or a tna, group_tna, tna_seq_data, tna_data, nestimate_data, netobject or stslist object. With action, a long event log (one row per event). NA and empty strings are missing values, not a state; to model missingness as a state, recode it first.
lag	Integer lag (default 1). Positive lags pair each event with the one lag steps later, negative lags with the one |lag| steps earlier. A transition-count matrix supports lag = 1 only.
engine	Name of a registered engine (default "classical"); the built-in engines are listed by list_lsa_engines().
alternative	Alternative hypothesis for the residual and kappa p-values: "two.sided" (default), "greater" or "less".
alpha	Significance level that flags a transition as significant (default 0.05).
loops	Keep self-transitions in the model (default TRUE). FALSE declares every self-transition a structural zero.
structural_zeros	Optional K x K 0/1 matrix in which 0 marks a transition that cannot occur; default NULL (none). Combined with loops = FALSE, the diagonal is also forbidden. Supported by the classical engine.
labels	Optional character vector of state names. Default NULL takes them from the data (sorted values of character input).
group	Optional grouping: one value per sequence, or, with a long event log, the name of a column that is constant within each sequence. Default NULL (one model).
actor, action, time, order, session	Column names for a long event log (all default NULL). Supplying action (the state column) switches to long-format input. Rows are grouped into sequences by actor, session, or both, ordered by order or else by time.
time_threshold	Gap in seconds that starts a new sequence when time is given and session is not (default 900).
custom_format	Optional strptime() format for the time column (default NULL).
is_unix_time	Read the time column as a Unix epoch (default FALSE).
unix_time_unit	Unit of the epoch: "seconds" (default), "milliseconds" or "microseconds".
params	Named list of engine-specific parameters (default list()).
...	Engine-specific parameters, merged into params. For the engine wrappers, any argument of lsa() other than engine.

Value

An object of class c("lsa", "cograph_network"). Read it with transitions() (one row per from -> to pair: from, to, lag, count, expected, prob, prob_col, adj_res, p, yules_q, kappa, kappa_z, kappa_p, lift, sign, significant, plus any engine-specific columns), nodes() (one row per state), tests() (one row per tablewise test), initial() (one row per state), transition_probabilities()

and summary() (see lsa-methods). The object also stores the per-cell quantities as K x K matrices, the canonical lsa_data() input and the parameters of the call, for the inference and plotting functions.

With group, an object of class c("lsa_group", "list"): one lsa fit per group level. The reading verbs return the same tables with a leading group column, and summary() returns one row per group.

Long-format input

A long event log is turned into sequences before the model is fitted: one sequence per actor, per session, or per actor-session pair, or a single sequence when neither is given. Events are ordered within a sequence by order, or else by time; when time is given without session, a gap longer than time_threshold seconds starts a new sequence. If the session identifier is already unique across actors, pass it as actor and leave session unset. Check the number of sequences reported by summary(fit) against the number expected.

Conditions

Errors carry the class lagdynamics_error and one of:

- lagdynamics_bad_input: invalid data, labels, group, structural-zero matrix or long-format column.
- lagdynamics_unknown_engine: engine is not registered.
- lagdynamics_missing_package: the engine needs a package that is not installed.
- lagdynamics_no_transitions: the data contain no transition at the requested lag.
- lagdynamics_unsupported_option: the engine does not support structural zeros.
- lagdynamics_ipf_infeasible: the structural zeros leave a state with observed transitions but no permitted cell.
- lagdynamics_needs_events: group was given with a transition-count matrix, which cannot be split.

A warning of class lagdynamics_structural_zero_counts is raised when the data contain transitions in cells declared structural zeros: the row and column totals then include transitions the model declares impossible, and the tablewise test and residuals are not interpretable.

References

Bakeman, R., & Gottman, J. M. (1997). *Observing interaction: An introduction to sequential analysis* (2nd ed.). Cambridge University Press.

Bakeman, R., & Quera, V. (1995). *Analyzing interaction: Sequential analysis with SDIS and GSEQ*. Cambridge University Press.

Christensen, R. (1997). *Log-linear models and logistic regression* (2nd ed.). Springer.

Haberman, S. J. (1979). *Analysis of qualitative data: Volume 2, New developments*. Academic Press.

Sackett, G. P. (1979). The lag sequential analysis of contingency and cyclicity in behavioral interaction research. In J. D. Osofsky (Ed.), *Handbook of infant development* (pp. 623-649). Wiley.

Wampold, B. E. (1984). Tests of dominance in sequential categorical data. *Psychological Bulletin, 96*(2), 424-429.

The formula-by-formula source map is installed with the package at `system.file("REFERENCES.md", package = "lagdynamics")`.

See Also

`transitions()`, `nodes()`, `tests()`, `initial()` to read the fit; `lsa_lags()` for several lags; `bootstrap_lsa()`, `permute_lsa()` and `compare_lsa()` for inference; `plot.lsa()`; `list_lsa_engines()` and `register_lsa_engine()` for engines; `lsa_data()` for the accepted input forms.

Examples

```
fit <- lsa(engagement)
fit
transitions(fit, significant = TRUE, sort = "strength")
tests(fit)

# The same model through an engine wrapper, at lag 2
lsa_classical(engagement, lag = 2)

# A long event log, one model per achievement group
gfit <- lsa(group_regulation_long, actor = "Actor", action = "Action",
            time = "Time", group = "Achiever")
summary(gfit)
```

---

lsa-inference-methods *Print and read resampling results*

Description

`print()` and `as.data.frame()` read the results of `bootstrap_lsa()`, `permute_lsa()` and `stability_lsa()`. `print()` shows a short summary: the engine, the number of replicates, the resampling settings and how many transitions are stable or significant (with the number of failed replicates for a stability result). `as.data.frame()` returns the per-transition table of the result.

`transitions()` returns the same table and adds the arguments `significant` (keep the transitions the method flags) and `sort` (order by the size of the effect); it is the usual way to read these results. Results of `certainty_lsa()` use the bootstrap methods.

Usage

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

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

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

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

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

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

Arguments

x	An lsa_bootstrap, lsa_permutation or lsa_stability object.
...	Not used.
row.names, optional	Not used; present for consistency with the generic.

Value

print() returns x invisibly. as.data.frame() returns a data frame with one row per transition (from -> to pair); its columns are listed under *Value* in bootstrap_lsa(), permute_lsa() and stability_lsa().

See Also

transitions(), bootstrap_lsa(), permute_lsa(), stability_lsa().

Examples

```
fit <- lsa(engagement)
st <- stability_lsa(fit, R = 20)
st
head(as.data.frame(st))
```

---

lsa-methods *Print, summarise and tidy a lag sequential analysis fit*

Description

print(), summary(), as.data.frame() and nobs() describe a fitted lag sequential analysis, an lsa object from lsa(). A fit has three levels of description: the model as a whole (its size and the tablewise test of independence), the states (how often each is left and entered, and how often a sequence starts in it), and the individual transitions. These methods give the first two levels; transitions() gives the third.

`print()` shows a compact overview: the engine and lag, the numbers of states, transitions, events and sequences, the likelihood-ratio test of independence, the strongest over-represented transitions and the initial-state distribution. `summary()` returns a `summary.lsa` object holding two tables, a one-row overview of the fit and one row per state, which are read with `as.data.frame(summary(fit), what = )`. For a grouped fit, `summary()` returns one overview row per group. `as.data.frame()` on a fit returns the full edge table, including the integer state ids and the network columns that `transitions()` leaves out. `nobs()` is the number of transitions the model was fitted to.

For the per-transition, per-state and per-test tables, prefer the reading verbs `transitions()`, `nodes()`, `tests()` and `initial()`.

Usage

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

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

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

## S3 method for class 'summary.lsa'
print(x, ...)

## S3 method for class 'summary.lsa'
as.data.frame(
  x,
  row.names = NULL,
  optional = FALSE,
  what = c("overview", "nodes"),
  ...
)

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

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

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

## S3 method for class 'lsa'
nobs(object, ...)
```

Arguments

`x, object`	An `lsa` fit from `lsa()`, or a grouped fit from `lsa(..., group = )`. For `print()` and `as.data.frame()` on a summary, a `summary.lsa` object.

...	Ignored.
row.names	Optional row names for the returned data frame; ignored by the grouped method.
optional	Ignored; present for consistency with the generic.
what	For as.data.frame() on a summary: "overview" (default, the one-row overview of the fit) or "nodes" (one row per state).

Value

- print() returns x invisibly.
- summary() on an lsa fit returns an object of class summary.lsa holding two tables. The overview has one row and the columns engine, lag, n_states, n_sequences, n_events, n_transitions, n_significant, alpha, lrx2, lrx2_df, lrx2_p, x2, x2_df, x2_p (NA where the engine computed no tablewise test or the input was a transition matrix). The node table has one row per state and the columns state, outgoing, incoming, out_share, in_share, init_prob. Read them with as.data.frame(s) and as.data.frame(s, what = "nodes").
- summary() on a grouped fit returns a data.frame with one row per group: a leading group column followed by the overview columns.
- as.data.frame() on an lsa fit returns a data.frame with one row per from -> to cell: integer from and to ids, from_label, to_label, the statistic columns of transitions(), and the network columns edge and weight. On a grouped fit the tables are stacked under a leading group column.
- as.data.frame() on a summary returns the overview or the node table.
- nobs() returns a single number, the total transition count.

See Also

lsa(); transitions(), nodes(), tests() and initial() for the reading verbs.

Examples

```
fit <- lsa(engagement)
fit
s <- summary(fit)
s
as.data.frame(s)
as.data.frame(s, what = "nodes")
nobs(fit)
head(as.data.frame(fit))

gfit <- lsa(group_regulation_long, actor = "Actor", action = "Action",
            time = "Time", group = "Achiever")
gfit
summary(gfit)
```

`lsa_data` *Convert sequence input to the canonical form used by lag sequential analysis*

Description

`lsa_data()` converts the forms in which sequence data usually arrive into the canonical representation used by lag sequential analysis, and it is what `lsa()` calls on its input. A lag sequential analysis needs, for every sequence, the ordered list of coded events, and a common set of state labels across sequences. Transitions are counted within sequences only, so the boundaries between sequences must be kept.

Accepted forms are:

- an atomic vector of codes (one sequence);
- a list of vectors (several sequences);
- a wide matrix or data frame, one row per sequence and one column per ordered time point;
- a square numeric matrix of transition counts, row `i` and column `j` holding the count of `i -> j` (any square numeric matrix is read this way, so pass square integer-coded sequences as a data frame);
- a sequence object: a `tna` or `group_tna` model with its data, a `tna_seq_data`, `tna_data` or `nestimate_data` object, a **Nestimate** `netobject`, or a **TraMineR** `stslist`.

`NA` and empty strings are treated as missing observations, not as a state: they are dropped wherever they occur and the surrounding events close up. To model missingness as a state, recode it (for example to `"missing"`) first. A transition-count matrix carries no sequences, so the functions that resample sequences (`bootstrap_lsa()`, `permute_lsa()`, `stability_lsa()`, `reliability_lsa()`, `compare_lsa()`) cannot be used on a fit built from it. Long event logs are read by `lsa()` through its `actor` and `action` arguments.

Usage

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

Arguments

`x`	Sequence input in one of the forms listed above. An `lsa_data` object is returned unchanged.
`labels`	Optional character vector of state names. Default `NULL` takes the sorted distinct values of character input, the levels of a factor, or `"Code 1"`, `"Code 2"`, ... for integer codes. When supplied, character codes are matched to it by name and integer codes index into it.

Value

An object of class `c("lsa_data", "list")`. Print it for the input source, the number of states, sequences, events and lag-1 transitions; read the events with `as.data.frame()` (one row per event, or per cell for a transition matrix). The object stores the integer-coded events and their sequence membership (`NULL` for a transition matrix), the state labels, the counts of states, sequences and events, and the source (`"events"` or `"transitions"`).

Conditions

Errors carry the class `lagdynamics_error` and `lagdynamics_bad_input`: an unsupported input type, no usable events, invalid counts in a transition matrix, invalid integer codes, or codes that do not match `labels`.

See Also

`as.data.frame.lsa_data()` for the event table; `lsa_transitions()` for transition counts; `lsa()`.

Examples

```
lsa_data(engagement)

# Several sequences: no transition is counted across the boundary
lsa_data(list(c("a", "b", "a"), c("b", "c", "a", "b")))

# A transition-count matrix
counts <- matrix(c(0, 3, 1, 2, 0, 4, 5, 1, 0), 3, 3,
                 dimnames = list(c("a", "b", "c"), c("a", "b", "c")))
lsa_data(counts)
```

`lsa_ipf` *Fit expected counts with structural zeros by iterative proportional fitting*

Description

`lsa_ipf()` fits the expected counts of a count table with structural zeros by iterative proportional fitting. When some transitions cannot occur by construction (structural zeros, such as self-transitions in a coding scheme that merges repeated codes), the quasi-independence model fits expected counts that are zero in the forbidden cells and reproduce the observed row and column totals; it has no closed form and is found by iterative proportional fitting (Wickens, 1989), which rescales the rows and columns of the table in turn until both sets of totals match. Under ordinary independence, the expected count of a transition is the product of its row and column totals divided by the grand total, a formula that assigns expected counts to impossible cells.

`lsa()` calls `lsa_ipf()` when `loops = FALSE` or `structural_zeros` is given, so it is needed directly only to inspect the expected table or to use it outside `lsa()`.

Usage

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

Arguments

obs	Square numeric matrix of observed transition counts.
structure	Square 0/1 matrix of the same size: 0 marks a structural zero, 1 an estimable cell. Default NULL: every cell is estimable, which gives the ordinary independence table.
tol	Convergence tolerance on the largest difference between fitted and observed totals (default 1e-8).
max_iter	Maximum number of row-and-column passes (default 200).

Value

A list with the fitted K x K expected-count matrix (fit), the number of passes used (iterations), whether the totals converged within max_iter (converged), and the largest remaining difference between fitted and observed totals (max_margin_diff).

Conditions

Errors carry the class lagdynamics_error and one of:

- lagdynamics_bad_input: obs contains missing, infinite or negative values.
- lagdynamics_ipf_infeasible: a row or column has observed transitions but every one of its cells is a structural zero.

References

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

See Also

lsa() and its loops and structural_zeros arguments.

Examples

```
obs <- matrix(c(0, 4, 6,
                3, 0, 5,
                7, 2, 0), nrow = 3, byrow = TRUE)
# Every cell estimable: the ordinary independence table
lsa_ipf(obs)

# Self-transitions forbidden: zero expected counts on the diagonal,
# observed row and column totals kept
lsa_ipf(obs, structure = 1 - diag(3))
```

lsa_lags *Fit lag sequential analysis models at several lags*

Description

`lsa_lags()` fits `lsa()` once per requested lag, with the same engine and options, and keeps the fits together. The result is a lag profile in the sense of Sackett (1979), who proposed examining a transition across a range of lags to separate short-range from longer-range dependence and to detect cycles: a sequential contingency need not be immediate, a state may tend to recur two or three events later, and an association that is strong at lag 1 may fade with distance.

`transitions()` on the result stacks the per-lag transition tables into one table with a `lag` column, with the same filtering and sorting arguments as for a single fit. To follow one transition across lags, use `lag_profile()`.

Usage

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

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

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

Arguments

`data`	Sequences in any form accepted by `lsa()`.
`lags`	Integer vector of lags (default `1:3`). Negative lags look back to predecessors.
`...`	Passed to `lsa()` (for example `engine`, `alpha`, `loops`). Ignored by the `print()` and `as.data.frame()` methods.
`x`	An `lsa_lags` object.
`row.names`	Optional row names for the stacked data frame.
`optional`	Ignored; present for consistency with the generic.

Value

An object of class `c("lsa_lags", "list")`, one `lsa` fit per lag. `transitions()` and `as.data.frame()` return a `data.frame` with one row per transition and lag and the columns of `transitions()` for a single fit. `print()` lists the number of significant transitions at each lag and returns `x` invisibly.

References

Sackett, G. P. (1979). The lag sequential analysis of contingency and cyclicity in behavioral interaction research. In J. D. Osofsky (Ed.), *Handbook of infant development* (pp. 623-649). Wiley.

See Also

lag_profile() for one transition across lags; lsa(); transitions().

Examples

```
prof <- lsa_lags(engagement, lags = 1:3)
prof
transitions(prof, direction = "over", sort = "strength")
```

---

lsa_to_tna *Convert a lag sequential analysis fit to a transition network for tna*

Description

lsa_to_tna() converts a lag sequential analysis fit to a transition network for the **tna** package: a tna network from an lsa fit, or a group_tna from a grouped fit, so that a network found by LSA can be analysed with the network tools of **tna**. Lag sequential analysis and Transition Network Analysis (TNA) read the same transitions from two angles. LSA tests which transitions are more or less frequent than independence predicts; TNA treats the transition probabilities as a weighted network and studies its structure with centralities, communities, cliques and resampling-based pruning.

The edge weights are chosen by weights: the transition probabilities (the TNA model), the counts, the positive adjusted residuals (the over-representation network) or the lift. With probabilities or counts the network also carries the initial-state probabilities and the sequences, so the resampling functions of **tna** can be used on it. With residuals or lift it carries neither, because **tna** would re-estimate probabilities from the sequences; resample with bootstrap_lsa() instead.

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 lsa(), or a grouped fit from lsa(..., group =).
...	Passed between methods.
weights	Edge weights: "prob" (default, transition probabilities), "count" (observed counts), "adj_res" (adjusted residuals, negative values set to 0) or "lift" (observed over expected counts).

Value

A `tna` object (for an `lsa` fit) or a `group_tna` object (one `tna` per group, for a grouped fit), read with the functions of the **tna** package.

Conditions

Errors carry the class `lagdynamics_error` and `lagdynamics_missing_package` when **tna** is not installed. With `weights = "adj_res"` or `"lift"` and a fit that has sequences, a warning of class `lagdynamics_tna_no_data` notes that the network carries no sequence data.

See Also

`transition_probabilities()` and `initial()` for the same parameters as tidy tables; `lsa()`.

Examples

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

---

lsa_transitions *Count transitions at a given lag*

Description

`lsa_transitions()` counts the transitions at a given lag and returns the table of observed transitions with its row and column totals: for each pair of states, how often the second occurs `lag` events after the first. This table is the starting point of every lag sequential analysis. At lag 1 it records immediate succession; at larger lags it records what tends to follow a state two or more events later, and at negative lags what tends to precede it. Transitions are counted within sequences only; no transition spans the boundary between two sequences.

The count involves no model or test; `lsa()` adds the expected counts, residuals and the other statistics. Use `lsa_transitions()` to inspect the raw counts, or to check how many transitions a lag leaves before fitting.

Usage

```
lsa_transitions(x, lag = 1)

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

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

Arguments

x	An lsa_data() object or any input it accepts.
lag	Integer lag (default 1). Positive lags count successors, negative lags predecessors. Input given as a transition-count matrix supports lag = 1 only.
row.names, optional, ...	Ignored; present for consistency with the generics.

Value

An object of class c("lsa_transitions", "list"). print() shows the lag, the number of states and transitions and the count matrix, and returns x invisibly. as.data.frame() returns one row per from -> to pair with the columns from, to, lag, count, row_total, col_total and n_transitions.

Conditions

Errors carry the class lagdynamics_error and lagdynamics_bad_input: a lag other than 1 for a transition-count matrix, or input that lsa_data() rejects.

See Also

lsa_data(), lsa(), lsa_lags()

Examples

```
tx <- lsa_transitions(engagement)
tx
as.data.frame(tx)

# What precedes each state
lsa_transitions(engagement, lag = -1)
```

---

nodes *Read the states of a lag sequential analysis fit*

Description

nodes() returns the states of a lag sequential analysis fit, one row per state with its total outgoing and incoming transition counts. In a lag transition network the states are the nodes, and how often a state is left and how often it is entered describe its place in the process: a state with many outgoing but few incoming transitions tends to open episodes, one with the reverse tends to close them. The marginal totals also set the expected counts against which every transition is tested.

The shares of all transitions and the initial-state probability are in as.data.frame(summary(fit), what = "nodes"); the per-transition table is transitions().

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(), or a grouped fit from lsa(..., group =).

Value

A data.frame with one row per state and the columns state (matching the from and to names of transitions()), outgoing and incoming. A grouped fit gives one row per group and state, with a leading group column.

See Also

transitions(), initial(), tests(), lsa-methods for summary().

Examples

```
nodes(lsa(engagement))
```

---

permute_lsa *Test transitions against no sequential structure by permutation*

Description

permute_lsa() tests every transition of a fit from lsa() against no sequential structure by permutation: whether the transition is over- or under-represented beyond what the same events in a random order would give, keeping the frequency of every state fixed. The adjusted residual of a transition is otherwise tested against a normal reference distribution, which holds only approximately, for large counts and independent transitions. A permutation test builds the reference distribution from the data instead: if the order of events carried no information, shuffling the events would produce residuals as large as the observed ones.

The events of the fit are shuffled many times, by default within each sequence so that every sequence keeps its own length and state frequencies; the model is refitted with the same engine and settings, and the adjusted residual of every transition is recorded. The two-sided permutation p-value of a transition is the share of shuffles whose residual is at least as large in absolute value as the observed one, counted so that it is never zero (Phipson & Smyth, 2010). A transition is significant when its p-value is below the alpha of the fit.

The p-values are not corrected for the number of transitions tested. For intervals on the residuals and other statistics, use `bootstrap_lsa()`; for the share of subsamples in which a transition stays significant, use `stability_lsa()`. To test whether groups differ, use `compare_lsa()`.

Usage

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

Arguments

`fit`	An `lsa` fit from `lsa()`, built from sequences (not from a transition matrix).
`R`	Number of permutations. Default `1000`.
`within_sequence`	Shuffle events within each sequence (`TRUE`, default) or across the whole event stream (`FALSE`). A single sequence is always shuffled as a whole, with a warning when `within_sequence = TRUE`.
`shuffles`	Optional list of at least `R` permutations of `seq_len(n_events)` to replay instead of random shuffles. Default `NULL`.
`parallel`	Run the permutations on several cores. Default `FALSE`.
`n_cores`	Number of cores when `parallel = TRUE`. Default `NULL`, all cores but one.
`verbose`	Report progress every 100 permutations. Default `FALSE`.
`...`	Not used.

Value

An object of class `c("lsa_permutation", "list")`. Read it with `transitions()`, which returns a data frame with one row per transition (`from -> to` pair) and the columns `from`, `to`, `observed_count`, `observed_adj_res`, `p_perm` (permutation p-value; `NA` for transitions that cannot be estimated, such as structural zeros) and `significant` (`p_perm` below the fit's `alpha`). `transitions(x, significant = TRUE)` keeps the significant transitions; `sort = "strength"` orders by the absolute observed residual. `print()` shows the settings and the number of significant transitions (see lsa-inference-methods). The object also stores the permuted residuals, the settings and the original fit.

Conditions

Errors carry the class `lagdynamics_error` and one of:

- `lagdynamics_needs_events`: the fit was built from a transition matrix, so there are no events to shuffle.
- `lagdynamics_bad_input`: `shuffles` is not a list of at least R permutations of `seq_len(n_events)`.

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

`transitions()` to read the result; `bootstrap_lsa()`, `stability_lsa()` and `reliability_lsa()` for other checks of a fit; `compare_lsa()` to compare groups.

Examples

```
fit <- lsa(engagement)
pm <- permute_lsa(fit, R = 100)
pm
transitions(pm, significant = TRUE, sort = "strength")

# A full analysis uses more permutations
permute_lsa(fit, R = 1000) |>
  transitions(sort = "strength")
```

---

plot.lsa *Plot a lag sequential analysis fit*

Description

`plot()` draws a lag sequential analysis fit as a residual heatmap, a lag transition network, a chord diagram or a polar sunburst, chosen with `type`; each display suits a different question. A residual heatmap lays out the full table of antecedent by consequent states, coloured by the adjusted residual, and shows at a glance which transitions are over-represented and which are avoided. A lag transition network draws the states as nodes and the transitions as directed edges, which shows the paths the process takes. A chord diagram shows how the transitions out of each state are shared among the targets, and a polar sunburst shows the same per-source breakdown on a ring.

The heatmap (default) shows the adjusted residuals, with significant cells in bold; `which` switches it to transition probabilities, observed counts or expected counts. The other types are drawn by `plot_transitions()`, `plot_chords()` and `plot_polar()`, whose arguments can be passed through `...`. A grouped fit is drawn once per group, or in one figure with `combined = TRUE`.

For the uncertainty of the residuals use `plot_forest()` on a bootstrap; for differences between groups, `plot()` on the result of `compare_lsa()` (see `plot.lsa_comparison()`).

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

x	An lsa fit from lsa(), or a grouped fit from lsa(..., group =).
type	"heatmap" (default), "network", "chord" or "sunburst".
...	Passed to the chosen display. For "heatmap": which, one of "residuals" (default), "prob", "count" or "expected". For the other types, the arguments of plot_transitions(), plot_chords() or plot_polar().
combined	For a grouped fit: FALSE (default) draws one figure per group; TRUE tiles the groups in one figure.

Value

For "heatmap" and "sunburst", a ggplot object, drawn when printed. For "network" and "chord", the value of plot_transitions() or plot_chords(), invisibly, with the plot drawn on the current device. For a grouped fit, a list with one such value per group, invisibly, after drawing.

Conditions

Errors carry the class lagdynamics_error and lagdynamics_bad_input when the fit holds no finite values of the quantity requested for the heatmap.

See Also

plot_transitions(), plot_chords(), plot_polar(), plot_forest(), plot.lsa_comparison(), transitions()

Examples

```
fit <- lsa(engagement)
plot(fit)
plot(fit, which = "prob")
plot(fit, type = "network")
plot(fit, type = "chord")

gfit <- lsa(group_regulation_long, actor = "Actor", action = "Action",
            time = "Time", group = "Achiever")
plot(gfit, combined = TRUE)
```

`plot.lsa_certainty` *Plot credible intervals of transition probabilities*

Description

`plot()` draws the result of `certainty_lsa()` as a circular forest plot: one spoke per transition, showing the observed transition probability and its credible interval, so the precisely and imprecisely estimated transitions can be seen side by side. The plot is drawn by `plot_forest()` with `metric = "prob"`; see that page for the layout and the arguments that can be passed through `...`.

Usage

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

Arguments

`x`	An `lsa_certainty` object from `certainty_lsa()`.
`metric`	Interval to draw. Default `"prob"`, the only interval a certainty result has.
`...`	Passed to `plot_forest()`.

Value

A `ggplot` object, drawn when printed.

See Also

`certainty_lsa()`, `plot_forest()`.

Examples

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

`plot.lsa_comparison` *Plot a comparison of transitions between groups*

Description

`plot()` draws the result of `compare_lsa()` or `bayes_compare_lsa()` as a barrel chart (default) or a heatmap. When the same states are observed in two or more groups, a comparison shows which transitions differ between them, by how much, and whether the difference is larger than chance, so a display of it has to show both the transitions themselves in each group and the test of their difference.

The barrel is a back-to-back chart with one row per transition: the first group's bar runs to the left and the second's to the right, bar length is the transition probability (or count), bar colour is the log odds ratio of the transition within the group (blue over-represented, red avoided), and the centre shows the p-value of the difference, in bold when significant. The rows are the `top_n` most frequent transitions, or with `rank = "effect"` those with the strongest association in either group. The heatmap draws the difference between the groups as a grid of `from` by `to` states. With more than two groups, each pair is drawn as a facet.

The tested differences are read as a table with `transitions(cmp)`. To colour the ribbons of a chord diagram by a group difference, see `plot_chords()`.

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

`x`	The result of `compare_lsa()` or `bayes_compare_lsa()`.
`style`	`"barrel"` (default) or `"heatmap"`.

value	For the barrel, bar length: "prob" (default) or "count".
rank	For the barrel, which transitions to show: "frequency" (default, the most frequent overall) or "effect" (the strongest association, largest absolute log odds ratio, in either group).
top_n	For the barrel, the number of transitions shown (default 12); with several pairs the same transitions are shown in every facet.
...	Ignored.

Value

A ggplot object, drawn when printed.

See Also

compare_lsa(), bayes_compare_lsa(), transitions(), plot_chords()

Examples

```
gfit <- lsa(group_regulation_long, actor = "Actor", action = "Action",
            time = "Time", group = "Achiever")
set.seed(1)
cmp <- compare_lsa(gfit, R = 100, adjust = "BH")
plot(cmp)
plot(cmp, rank = "effect")
plot(cmp, style = "heatmap")
```

---

plot_chords *Plot a chord diagram of transitions*

Description

plot_chords() draws the transitions of a fit as a chord diagram with **cograph**: the states are arcs around a circle and each transition is a ribbon from its source arc to its target arc. The width of a ribbon shows how much of the traffic the transition carries, and its colour can show whether the transition is more or less frequent than independence predicts, so the diagram shows where the transitions out of each state go, and which of the frequent routes are also over-represented.

Ribbon width is the count (default) or the transition probability; ribbon colour is the adjusted residual on a diverging scale (red over-represented, blue avoided), or the lift, probability or count on a sequential scale. With compare, a second fit on the same states, each ribbon is coloured by the difference between the two fits in the colour quantity, which shows where two groups differ.

The same display is plot(fit, type = "chord"). For the full table use the residual heatmap of plot.lsa(); for a network layout, plot_transitions(); for a test of group differences, compare_lsa().

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

fit	An lsa fit from lsa().
compare	Optional second lsa fit with the same states; ribbons are then coloured by fit minus compare. Default NULL.
width	Ribbon width: "count" (default) or "prob".
color	Ribbon colour: "residuals" (default), "lift", "prob" or "count".
significant	Draw only transitions whose residual p-value is below the fit's alpha (default FALSE); ignored with compare.
self_loops	Draw self-transitions (default TRUE).
alpha	Ribbon opacity (default 0.6).
...	Passed to cograph::plot_chord(), for example ticks, segment_width, label_size or title.

Value

The value of cograph::plot_chord() (the drawn segments and chords), invisibly; the plot is drawn on the current device.

Conditions

Errors carry the class lagdynamics_error and lagdynamics_incompatible_fits when fit and compare have different states.

See Also

plot.lsa(), plot_transitions(), plot_polar(), compare_lsa()

Examples

```
fit <- lsa(engagement)
plot_chords(fit)
plot_chords(fit, width = "prob", self_loops = FALSE)

# Ribbons coloured by the difference between two groups
```

```
high <- lsa(subset(group_regulation_long, Achiever == "High"),
            actor = "Actor", action = "Action", time = "Time")
low <- lsa(subset(group_regulation_long, Achiever == "Low"),
           actor = "Actor", action = "Action", time = "Time")
plot_chords(high, compare = low)
```

`plot_forest` *Plot a circular forest of bootstrap confidence intervals*

Description

`plot_forest()` draws the bootstrap confidence intervals of a fit's transitions as a circular forest plot, one spoke per transition: the spoke spans the bootstrap confidence interval, a square marks the observed estimate, and a dashed ring marks the null value. A forest plot shows an estimate and its confidence interval for each of many quantities, so that their size and precision can be compared at once; here the quantities are the transitions, and the plot shows which transitions keep their direction across resamples of the sequences and how precisely each is estimated.

Transitions whose adjusted residual is stable across resamples are coloured by direction (red over-represented, blue avoided) and the others are grey. The interval can be that of the adjusted residual (default), the count, the transition probability or Yule's Q. `plot()` on a bootstrap result draws the same display.

The intervals come from `bootstrap_lsa()`; read them as a table with `transitions(boot)`. For analytic intervals of the transition probabilities, see `certainty_lsa()` and its plot method.

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`	A bootstrap result from `bootstrap_lsa()`.
`metric`	Interval to draw: `"residuals"` (default), `"count"`, `"prob"` or `"yules_q"`.
`n_top`	Keep only the `n_top` transitions with the largest absolute estimate. Default `NULL` (all).
`show_nonsig`	Draw transitions whose residual is not stable, in grey (default `TRUE`).
`label_size`	Size of the transition labels (default `2.6`).

x	A bootstrap result from bootstrap_lsa(), for plot().
...	Passed to plot_forest(), for example metric or n_top.

Value

A ggplot object, drawn when printed. plot() on a bootstrap result returns the same object.

Conditions

Errors carry the class lagdynamics_error and lagdynamics_bad_input when the bootstrap holds no interval for metric, or when no transition is left to draw.

See Also

bootstrap_lsa(), transitions(), plot.lsa(), certainty_lsa()

Examples

```
set.seed(1)
boot <- bootstrap_lsa(lsa(engagement), R = 100)
plot_forest(boot)
plot(boot, metric = "prob")

boot9 <- bootstrap_lsa(lsa(group_regulation), R = 500)
plot_forest(boot9, n_top = 20, show_nonsig = FALSE)
```

---

plot_polar *Plot a polar sunburst of transitions*

Description

plot_polar() draws the transitions of a fit as a polar sunburst with **ggplot2**: the source states are arranged around an inner ring and, outside each source, the transitions leaving it. It reads like a set of pie charts joined into one figure: for every state, where the process goes next and how often, and, through the fill colour, which of these moves are more or less frequent than independence predicts.

The sunburst has two styles. In the rose style (default) every target has a slot of equal angle within its source and the bar height shows the count or the probability, so that every target can be labelled. In the wedge style the angle of each wedge is the transition's share of its source's outgoing transitions, and very small wedges are dropped. The fill is the adjusted residual on the diverging scale of the residual heatmap (red over-represented, blue avoided), or the probability or lift.

The same display is plot(fit, type = "sunburst"). For the full table use plot.lsa(); for a network layout, plot_transitions(); for ribbons between states, plot_chords().

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

fit	An lsa fit from lsa().
style	"rose" (default) or "wedge".
fill	Fill quantity: "residuals" (default), "prob" or "lift".
size	For the rose style, bar height: "count" (default) or "prob".
significant	Grey out transitions whose residual p-value is not below the fit's alpha (default FALSE).
labels	Which targets to name: "all", "auto" (those with room) or "none". Default "all" for the rose and "auto" for the wedge style. Source names are always shown.
min_show	For the wedge style, drop wedges below this share of their source's outgoing transitions (default 0.01).
label_size	Label text size (default 3).
...	Ignored; allows plot(fit, type = "sunburst", ...).

Value

A ggplot object, drawn when printed.

Conditions

Errors carry the class lagdynamics_error and lagdynamics_bad_input when the fit has no transitions to draw.

See Also

plot.lsa(), plot_chords(), plot_transitions()

Examples

```
fit <- lsa(group_regulation)
plot_polar(fit)
plot_polar(fit, style = "wedge", significant = TRUE)
```

plot_transitions *Plot a lag transition network*

Description

plot_transitions() draws a lag transition network with **cograph**: the states are nodes and the transitions between them directed edges. Which quantity weights the edges decides what the network shows. Weighted by adjusted residuals it is a residual network: edges show where a transition is more frequent (over-represented) or less frequent (avoided) than independence predicts, the reading of lag sequential analysis. Weighted by transition probabilities it is the network of Transition Network Analysis, which shows where the process goes next, whether or not that is more than chance.

With weights = "residuals" (default), over-represented transitions are solid blue edges and avoided transitions dashed red edges; "yules_q" draws the same signed network on the bounded Yule's Q scale, and "lift" the ratio of observed to expected counts. With "prob" (or its aliases "tna" and "relative") or "count", the network is drawn in the style of Transition Network Analysis, each node ringed by its initial-state probability; probability edges below 0.05 are hidden unless edge_cutoff is passed. Dense networks are thinned with significant or, better at large sample sizes where nearly every cell is significant, with top.

The same display is plot(fit, type = "network"). For the full table use the residual heatmap of plot.lsa(); for the per-source breakdown, plot_chords() or plot_polar().

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

fit	An lsa fit from lsa().
weights	Edge weight: "residuals" (default), "tna", "relative", "prob" (the three are the transition probability), "count", "lift" or "yules_q".
significant	Keep only transitions whose residual p-value is below the fit's alpha (default FALSE).
top	Keep only the strongest edges by absolute weight: a fraction below 1 keeps that share of the edges, a number of 1 or more keeps that many. Default NULL (all edges). Applied after significant.

decimals	Decimal places of the edge labels (default 1).
node_fill	Node colour (default "white"). The probability and count networks colour nodes by state unless node_fill is set.
edge_labels	Show the edge weights as labels (default TRUE).
...	Passed to cograph::splot(), for example layout, node_shape, edge_cutoff or curvature.

Value

The network drawn by cograph::splot(), invisibly; the plot is drawn on the current device.

See Also

plot.lsa(), plot_chords(), transitions(), transition_probabilities()

Examples

```
fit <- lsa(engagement)
plot_transitions(fit)
plot_transitions(fit, weights = "prob")

fit9 <- lsa(group_regulation)
plot_transitions(fit9, top = 12)
plot_transitions(fit9, weights = "yules_q", significant = TRUE)
```

---

register_lsa_engine *Register a lag sequential analysis engine*

Description

register_lsa_engine() adds a lag sequential analysis engine under a name, after which lsa(data, engine = name) fits it; the fit is read, resampled and plotted like any other. An engine is the part of a lag sequential analysis that turns a table of observed transitions into expected counts and cell-wise tests. The built-in engines implement the classical analysis of Bakeman and Quera (1995), a two-cell test, Sackett's (1979) bidirectional test, and the parallel and non-parallel dominance tests (Wampold, 1984). Other tests of the same table, such as a different residual or a model-based expectation, can be added as engines and then used with every function of the package.

Re-registering a name replaces the engine for the session. The built-in engines are registered when the package loads.

Usage

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

Arguments

name	Name of the engine, used as lsa(engine = name).
fn	The engine function. It is called with transitions (the lsa_transitions() object), structural_zeros (NULL or a K x K 0/1 matrix), alternative, n_events (NULL for transition-matrix input) and any engine parameters given to lsa(), so it should accept transitions and .... It must return a named list holding at least the K x K matrices obs, exp, prob, adj_res and p; further K x K matrices become extra columns of transitions().
description	One-line description shown by list_lsa_engines().
requires	Names of packages the engine needs (default none); lsa() checks that they are installed before fitting.

Value

name, invisibly.

Conditions

Errors carry the class lagdynamics_error and lagdynamics_bad_input when fn has no transitions argument.

References

Bakeman, R., & Quera, V. (1995). *Analyzing interaction: Sequential analysis with SDIS and GSEQ*. Cambridge University Press.

Sackett, G. P. (1979). The lag sequential analysis of contingency and cyclicity in behavioral interaction research. In J. D. Osofsky (Ed.), *Handbook of infant development* (pp. 623-649). Wiley.

Wampold, B. E. (1984). Tests of dominance in sequential categorical data. *Psychological Bulletin, 96*(2), 424-429.

See Also

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

Examples

```
# An engine that tests Pearson residuals under independence
pearson_engine <- function(transitions, ...) {
  obs <- transitions$obs
  expected <- outer(rowSums(obs), colSums(obs)) / sum(obs)
  residual <- (obs - expected) / sqrt(expected)
  list(obs = obs, exp = expected, prob = obs / rowSums(obs),
       adj_res = residual, p = 2 * stats::pnorm(-abs(residual)))
}
register_lsa_engine("pearson", pearson_engine,
                    description = "Pearson residuals under independence")
list_lsa_engines()
fit <- lsa(engagement, engine = "pearson")
transitions(fit, significant = TRUE)
```

```
unregister_lsa_engine("pearson")
```

`reliability_lsa` *Estimate the split-half reliability of a lag transition network*

Description

`reliability_lsa()` estimates the split-half reliability of a lag transition network from one data set. A lag transition network is reliable when two independent samples of the same population give similar networks. Split-half reliability follows the resampling approach that Epskamp, Borsboom and Fried (2018) describe for the accuracy of psychological networks: the sequences are divided at random into two halves, a network is estimated on each half, and the two networks are compared transition by transition. A high correlation between the halves means that the structure of the network does not depend on which sequences happened to be sampled.

The random split is repeated `R` times. Each split assigns half of the sequences to one half and the rest to the other, refits the model on each with the same engine and settings, and correlates the two halves' transition probabilities, counts or adjusted residuals over all transitions. `reliability_lsa()` reports the mean, standard deviation and 95% interval (2.5% and 97.5% quantiles) of the split-half correlations, and the number of failed replicates, in which a half had no transitions or a half's values did not vary. A grouped fit gives one result per group.

Reliability describes the network as a whole. For the precision of each transition use `bootstrap_lsa()` or `certainty_lsa()`; for whether each transition stays significant when sequences are dropped, use `stability_lsa()`.

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, ...)

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

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

Arguments

fit	An lsa fit from lsa(), or a grouped fit from lsa(..., group =), built from at least two sequences (not from a transition matrix).
...	For a grouped fit, the arguments above, passed on to each group; otherwise not used.
R	Number of random splits. Default 100.
weights	Transition values correlated between the halves: "prob" (default, transition probabilities), "count" or "adj_res" (adjusted residuals).
method	Correlation: "pearson" (default) or "spearman".
parallel	Run the splits on several cores. Default FALSE.
n_cores	Number of cores when parallel = TRUE. Default NULL, all cores but one.
verbose	Report progress every 100 splits. Default FALSE.
x	An lsa_reliability or lsa_reliability_group object, for the print() methods.

Value

An object of class c("lsa_reliability", "list"). Read it with as.data.frame(): as.data.frame(x) gives one row per split with its correlation (NA for a failed replicate), and as.data.frame(x, what = "summary") one row with the settings, the number of failed replicates n_failed, and the mean, sd, ci_low and ci_high of the correlations. print() shows the same summary and returns x invisibly. For a grouped fit, an object of class c("lsa_reliability_group", "list") with one result per group, whose tables gain a leading group column.

Conditions

Errors carry the class lagdynamics_error and one of:

- lagdynamics_needs_events: the fit was built from a transition matrix, so there are no sequences to split.
- lagdynamics_bad_input: the fit has fewer than two sequences.

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.

See Also

as.data.frame.lsa_reliability() to read the result; bootstrap_lsa(), certainty_lsa(), stability_lsa() and permute_lsa() for transition-level checks.

Examples

```
fit <- lsa(engagement)
rel <- reliability_lsa(fit, R = 50)
rel
as.data.frame(rel, what = "summary")

# One result per group
g <- lsa(group_regulation_long, actor = "Actor", action = "Action",
         time = "Time", group = "Achiever")
reliability_lsa(g, R = 20) |>
  as.data.frame(what = "summary")

# A full analysis uses more splits
reliability_lsa(fit, R = 500)
```

`stability_lsa` *Assess the case-drop stability of significant transitions*

Description

`stability_lsa()` assesses the case-drop stability of the significant transitions of a fit from `lsa()`: whether a transition's significance survives when part of the data is left out, since a transition found significant in the full data may owe its significance to a few sequences. It follows the case-dropping approach that Epskamp, Borsboom and Fried (2018) describe for the stability of psychological networks: subsets of the cases are drawn without replacement, the model is refitted on each, and the share of subsets in which a transition is significant measures how much its significance depends on the particular cases sampled.

`R` subsets are drawn, each keeping a share `proportion` of the sequences (or, for a single sequence, a run of consecutive events of that share of its length); the model is refitted with the same engine and settings, and whether each transition is significant at the fit's `alpha` is recorded. A transition's stability is the share of subsets in which it was significant; it is flagged stable when that share reaches `min_stable`. Replicates in which a subset contained no transitions are reported as failed and count as not significant.

Stability concerns the significance decision, in either direction, not the size of the effect. For intervals on the statistics use `bootstrap_lsa()`; for a p-value against no sequential structure use `permute_lsa()`; for the agreement of the whole network between halves of the data use `reliability_lsa()`.

Usage

```
stability_lsa(
  fit,
  R = 500L,
  proportion = 0.8,
  min_stable = 0.95,
```

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

Arguments

`fit`	An `lsa` fit from `lsa()`, built from sequences (not from a transition matrix).
`R`	Number of subsets. Default `500`.
`proportion`	Share of the sequences kept in each subset, between 0 and 1. Default `0.8`.
`min_stable`	Share of subsets in which a transition must be significant to be flagged stable, greater than 0 and at most 1. Default `0.95`.
`parallel`	Run the subsets on several cores. Default `FALSE`.
`n_cores`	Number of cores when `parallel = TRUE`. Default `NULL`, all cores but one.
`verbose`	Report progress every 100 subsets. Default `FALSE`.
`...`	Not used.

Value

An object of class `c("lsa_stability", "list")`. Read it with `transitions()`, which returns a data frame with one row per transition (`from -> to` pair) and the columns `from`, `to`, `observed_sig` (significant in the full fit), `stability` (share of subsets in which it was significant) and `stable` (`stability >= min_stable`). `transitions(x, significant = TRUE)` keeps the stable transitions; `sort = "strength"` orders by stability. `print()` shows the settings, the number of failed replicates and the number of stable transitions (see lsa-inference-methods). The object also stores the per-subset significance decisions, the settings and the original fit.

Conditions

Errors carry the class `lagdynamics_error` and one of:

- `lagdynamics_needs_events`: the fit was built from a transition matrix, so there are no sequences to drop.

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.

See Also

`transitions()` to read the result; `bootstrap_lsa()`, `permute_lsa()` and `reliability_lsa()` for other checks of a fit.

Examples

```
fit <- lsa(engagement)
st <- stability_lsa(fit, R = 100)
st
transitions(st, significant = TRUE)

# A full analysis uses more subsets
stability_lsa(fit, R = 500, proportion = 0.5) |>
  transitions(sort = "strength")
```

`tests` *Read the tablewise tests of independence of a fit*

Description

`tests()` returns the tablewise tests of independence of a fit, one row per test. They test whether the table as a whole departs from independence: whether the state at one time carries any information about the state `lag` events later (Bakeman & Quera, 1995). The likelihood-ratio statistic G^2 and the Pearson χ^2 statistic test this omnibus hypothesis against a chi-square distribution, and are read before individual transitions are interpreted.

The tests are computed by the classical engine; the other built-in engines test cells only, and `tests()` then returns a table with no rows.

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()`, or a grouped fit from `lsa(..., group = )`.

Value

A `data.frame` with one row per test and the columns `test` (`"lrx2"` for the likelihood-ratio test, `"x2"` for Pearson's test), `statistic`, `df` and `p`. A grouped fit gains a leading `group` column.

References

Bakeman, R., & Quera, V. (1995). *Analyzing interaction: Sequential analysis with SDIS and GSEQ*. Cambridge University Press.

See Also

transitions() for the cell-wise tests; nodes(); lsa-methods for summary().

Examples

```
tests(lsa(engagement))
```

---

transfer_entropy *Estimate directed transfer entropy between categorical sequences*

Description

transfer_entropy() (**experimental**) estimates directed transfer entropy between categorical sequences. Transfer entropy (Schreiber, 2000) measures directed predictive information: how much knowing the past of a source series reduces uncertainty about the future of a target series, beyond what the target's own past already tells. Because it conditions on the target's history, it is not inflated when a state merely tends to persist, which inflates lagged association measures for processes with strong momentum. Transfer entropy is unsigned: a large value means strong directed dependence, which may make the target more or less likely. Read it alongside the signed residuals and Yule's Q of transitions().

It works in two modes. With one set of sequences (y = NULL), it treats each state as an indicator series (in the state or not) and returns the transfer entropy from every state to every other state, a directed information-flow network among the states. With very few states the indicator series of the other states carry nearly the same information, so several pairs can share a value; the bivariate mode avoids this. With two aligned sets of sequences (x and y), it returns the transfer entropy between the two series in both directions, using their full alphabets.

Transfer entropy is in bits. The surrogate test (default) shuffles the source to build a null distribution, which gives a p-value and an effective transfer entropy with the average surrogate value subtracted. For the classical test of sequential dependence use lsa().

Usage

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

Arguments

x	Categorical sequences: a vector (one sequence) or a matrix or data frame with one sequence per row and one time point per column, padded with NA, as accepted by lsa().
y	Optional second set of sequences of the same shape as x, for the bivariate mode. Default NULL (state-flow network of x).
lag	Prediction horizon, an integer of at least 1 (default 1).
history	Number of past steps of the target conditioned on, an integer of at least 1 (default 1).
test	"surrogate" (default) for the surrogate test, or "none".
R	Number of surrogates (default 199).
normalize	Add te_normalised, transfer entropy as a share of the target's remaining uncertainty given its own history, in [0, 1] (default TRUE).
seed	Optional seed for the surrogate test (default NULL). The session's random number state is restored on exit.

Value

A data.frame with one row per ordered pair (from, to) and the columns te (bits), te_effective (with test = "surrogate"), te_normalised (with normalize = TRUE), p (with test = "surrogate") and n (number of transitions used), sorted by decreasing te. In the state-flow mode there is one row per ordered pair of distinct states; in the bivariate mode two rows, named after the arguments.

Conditions

Errors carry the class lagdynamics_error and lagdynamics_bad_input when lag, history or R is not a positive whole number, or when x and y are not aligned (different numbers of sequences or different lengths within a sequence).

References

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

See Also

lsa(), transitions(), lsa_lags()

Examples

```
# Information flow among the engagement states
transfer_entropy(engagement, test = "none")

# With the surrogate test (an analysis would use more surrogates)
transfer_entropy(engagement, R = 19, seed = 1)

# Between two aligned series
mood <- c("calm", "calm", "tense", "tense", "calm", "tense", "tense", "calm")
effort <- c("low", "low", "low", "high", "high", "low", "high", "high")
```

```
transfer_entropy(mood, effort, test = "none")
```

`transitions` *Read the transitions of a fit or an inference result*

Description

`transitions()` returns the transitions of a lag sequential analysis as a table, one row per ordered pair of states, from a fit and from every result built on a fit. The transitions are the substance of the analysis: for each ordered pair of states, how often the second follows the first at the chosen lag, how often independence would predict, and whether the difference is larger than chance. A positive adjusted residual marks an over-represented transition, one the process prefers; a negative residual marks an avoided one. Yule's Q, kappa and lift express the same association on bounded or ratio scales.

On a fit from `lsa()`, a grouped fit or a multi-lag fit from `lsa_lags()`, it returns the estimated statistics of each transition. On the result of `bootstrap_lsa()`, `certainty_lsa()`, `stability_lsa()`, `permute_lsa()`, `compare_lsa()` or `bayes_compare_lsa()`, it returns that method's per-transition quantities. The arguments narrow and order the table: `significant = TRUE` keeps the transitions the method flags (significant for a fit, a permutation test or a group comparison; stable under resampling for the bootstrap, certainty and stability results), `direction` keeps over- or under-represented transitions, `min_count` drops rare ones, and `sort` puts the strongest first.

Use `nodes()`, `tests()` and `initial()` for the other tables of a fit, and `lag_profile()` to follow one transition across lags.

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, a grouped fit, an lsa_lags result, or the result of bootstrap_lsa(), certainty_lsa(), stability_lsa(), permute_lsa(), compare_lsa() or bayes_compare_lsa().
...	Arguments passed to the method.
significant	Keep only the transitions the method flags as significant or stable (default FALSE).
direction	For fits only: "any" (default), "over" (significant with a positive residual) or "under" (significant with a negative residual). "over" and "under" imply significant = TRUE.
min_count	For fits only: keep transitions observed at least this many times. Default NULL (no filter).
alpha	For fits only: significance level. Default NULL uses the alpha of the fit.

`sort` Row order: `"none"` (default, table order) or `"strength"` (largest absolute effect first: the adjusted residual for a fit, a bootstrap, certainty or permutation result; the stability proportion for `stability_lsa()`; the group difference for a comparison). Fits also accept `"count"` and `"prob"` (largest first).

Value

A `data.frame` with one row per transition. For a fit the columns are `from`, `to`, `lag`, `count`, `expected`, `prob` (row conditional, $P(\text{to} \mid \text{from})$), `prob_col` (column conditional, $P(\text{from} \mid \text{to})$), `adj_res`, `p`, `yules_q`, `kappa`, `kappa_z`, `kappa_p`, `lift` (observed over expected), `sign` (`"over"`, `"under"` or `"expected"`, comparing the count with the expected count) and `significant`; engines that compute further statistics append them (the two-cell engine adds `odds_ratio`, `log_or` and `log_or_se`). For an inference result the columns are `from`, `to` and that method's quantities, described on its help page; a pairwise comparison adds `group_a` and `group_b`. A grouped fit gains a leading `group` column, an `lsa_lags` result one row per transition and lag.

See Also

`lsa()`, `nodes()`, `tests()`, `initial()`, `transition_probabilities()`, `lag_profile()`

Examples

```
fit <- lsa(engagement)
transitions(fit)
transitions(fit, significant = TRUE, sort = "strength")
transitions(fit, direction = "under")
transitions(fit, min_count = 150, sort = "count")

# The same verb reads an inference result
transitions(certainty_lsa(fit), significant = TRUE)
```

`transition_probabilities` *Read the transition probabilities of a fit*

Description

`transition_probabilities()` returns the transition probabilities of a fit, one row per ordered pair of states. The transition probability $P(\text{to} \mid \text{from})$ is the share of the transitions leaving a state that go to each other state, so the probabilities out of one state sum to 1. They describe where the process goes next, regardless of chance expectation; the adjusted residuals of `transitions()` say which of these moves are more or less frequent than independence predicts. Together with the initial-state probabilities of `initial()`, they are the parameters of the first-order Markov model read by Transition Network Analysis.

A cell declared a structural zero has probability 0, as does every cell of a state that is never left.

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(), or a grouped fit from lsa(..., group =).

Value

A data.frame with one row per from -> to pair (K^2 rows for K states, ordered by from and then to) and the columns from, to and prob. A grouped fit stacks the groups under a leading group column.

See Also

initial() for the initial-state probabilities; transitions(), which also carries prob; lsa_to_tna().

Examples

```
transition_probabilities(lsa(engagement))
```

---

unregister_lsa_engine *Remove a registered engine*

Description

unregister_lsa_engine() removes an engine from the registry for the rest of the session, so that lsa(engine = name) no longer finds it. It is mainly used to clean up an engine added with register_lsa_engine(). A removed built-in engine returns when the package is loaded again.

Usage

```
unregister_lsa_engine(name)
```

Arguments

name	Name of the engine.

Value

NULL, invisibly.

Conditions

Errors carry the class `lagdynamics_error` and `lagdynamics_unknown_engine` when no engine is registered under `name`.

See Also

`register_lsa_engine()`, `list_lsa_engines()`

Examples

```
placeholder <- function(transitions, ...) {
  obs <- transitions$obs
  expected <- outer(rowSums(obs), colSums(obs)) / sum(obs)
  list(obs = obs, exp = expected, prob = obs / rowSums(obs),
       adj_res = obs * 0, p = obs * 0 + 1)
}
register_lsa_engine("placeholder", placeholder,
                    description = "Placeholder engine")
unregister_lsa_engine("placeholder")
list_lsa_engines()
```

Index

∗ **datasets**
ai_long, 3
engagement, 18
group_regulation, 20
group_regulation_long, 20

ai_long, 3
as.data.frame(), *54*
as.data.frame.lsa *(*lsa-methods*)*, 29
as.data.frame.lsa_bootstrap *(*lsa-inference-methods*)*, 28
as.data.frame.lsa_certainty_group *(*certainty_lsa*)*, 13
as.data.frame.lsa_comparison, 4
as.data.frame.lsa_comparison(), *9, 10*, *17*
as.data.frame.lsa_comparison_pairwise *(*as.data.frame.lsa_comparison*)*, 4
as.data.frame.lsa_data, 5
as.data.frame.lsa_data(), *33*
as.data.frame.lsa_group *(*lsa-methods*)*, 29
as.data.frame.lsa_lags *(*lsa_lags*)*, 35
as.data.frame.lsa_permutation *(*lsa-inference-methods*)*, 28
as.data.frame.lsa_reliability, 6
as.data.frame.lsa_reliability(), *54*
as.data.frame.lsa_reliability_group *(*as.data.frame.lsa_reliability*)*, 6
as.data.frame.lsa_stability *(*lsa-inference-methods*)*, 28
as.data.frame.lsa_transitions *(*lsa_transitions*)*, 37
as.data.frame.summary.lsa *(*lsa-methods*)*, 29

bayes_compare_lsa, 8
bayes_compare_lsa(), *4, 5*, *13–15*, *17*, *44, 45*, *60, 61*
bootstrap_lsa, 10
bootstrap_lsa(), *13, 14*, *25*, *28, 29*, *32*, *36*, *40, 41*, *47, 48*, *53–56*, *60, 61*

certainty_lsa, 13
certainty_lsa(), *8*, *10–12*, *25*, *28*, *43*, *47, 48*, *53, 54*, *60, 61*
cograph::plot_chord(), *46*
cograph::splot(), *51*
compare_lsa, 15
compare_lsa(), *4, 5*, *8*, *10*, *12*, *21*, *25*, *28*, *32*, *40, 41*, *44–46*, *60, 61*

engagement, 18, *20*

get_lsa_engine, 19
get_lsa_engine(), *24*, *52*
group_regulation, *18*, 20, *21*
group_regulation_long, *3*, *20*, 20

initial, 22
initial(), *25, 26*, *28*, *30, 31*, *37*, *39*, *60*, *62, 63*

lag_profile, 23
lag_profile(), *35, 36*, *60*, *62*
list_lsa_engines, 24
list_lsa_engines(), *19*, *26*, *28*, *52*, *64*
lsa, 24
lsa(), *3*, *10, 11*, *13, 14*, *17–24*, *29–40*, *42*, *46*, *49, 50*, *52*, *54–60*, *62, 63*
lsa-inference-methods, *12*, 28, *40*, *56*
lsa-methods, *27*, 29, *39*, *58*
lsa_bidirectional *(*lsa*)*, 24
lsa_classical *(*lsa*)*, 24
lsa_data, 32
lsa_data(), *5, 6*, *26–28*, *38*
lsa_ipf, 33
lsa_lags, 35

lsa_lags(), *23*, *25*, *28*, *38*, *59, 60*
lsa_nonparallel_dominance *(*lsa*)*, 24
lsa_parallel_dominance *(*lsa*)*, 24
lsa_to_tna, 36
lsa_to_tna(), *63*
lsa_transitions, 37
lsa_transitions(), *6*, *33*, *52*
lsa_two_cell *(*lsa*)*, 24

nobs.lsa *(*lsa-methods*)*, 29
nodes, 38
nodes(), *22*, *25, 26*, *28*, *30, 31*, *58*, *60*, *62*

permute_lsa, 39
permute_lsa(), *11, 12*, *17*, *25*, *28, 29*, *32*, *54–56*, *60, 61*
plot(), *9*, *17*
plot.lsa, 41
plot.lsa(), *25*, *28*, *45, 46*, *48–51*
plot.lsa_bootstrap *(*plot_forest*)*, 47
plot.lsa_certainty, 43
plot.lsa_certainty(), *14*
plot.lsa_comparison, 44
plot.lsa_comparison(), *10*, *17*, *41, 42*
plot.lsa_comparison_pairwise *(*plot.lsa_comparison*)*, 44
plot.lsa_group *(*plot.lsa*)*, 41
plot_chords, 45
plot_chords(), *41, 42*, *44, 45*, *48–51*
plot_forest, 47
plot_forest(), *12*, *14*, *41–43*
plot_polar, 48
plot_polar(), *41, 42*, *46*, *50*
plot_transitions, 50
plot_transitions(), *41, 42*, *45, 46*, *48, 49*
print.lsa *(*lsa-methods*)*, 29
print.lsa_bayes *(*bayes_compare_lsa*)*, 8
print.lsa_bayes_pairwise *(*bayes_compare_lsa*)*, 8
print.lsa_bootstrap *(*lsa-inference-methods*)*, 28
print.lsa_certainty *(*certainty_lsa*)*, 13
print.lsa_certainty_group *(*certainty_lsa*)*, 13
print.lsa_comparison *(*as.data.frame.lsa_comparison*)*, 4
print.lsa_comparison_pairwise *(*as.data.frame.lsa_comparison*)*, 4
print.lsa_data *(*as.data.frame.lsa_data*)*, 5
print.lsa_group *(*lsa-methods*)*, 29
print.lsa_lags *(*lsa_lags*)*, 35
print.lsa_permutation *(*lsa-inference-methods*)*, 28
print.lsa_reliability *(*reliability_lsa*)*, 53
print.lsa_reliability_group *(*reliability_lsa*)*, 53
print.lsa_stability *(*lsa-inference-methods*)*, 28
print.lsa_transitions *(*lsa_transitions*)*, 37
print.summary.lsa *(*lsa-methods*)*, 29

register_lsa_engine, 51
register_lsa_engine(), *19*, *24*, *28*, *63, 64*
reliability_lsa, 53
reliability_lsa(), *6, 7*, *12*, *32*, *41*, *55, 56*

stability_lsa, 55
stability_lsa(), *11, 12*, *28, 29*, *32*, *40, 41*, *53, 54*, *60–62*
stats::p.adjust(), *9, 10*, *16, 17*
summary.lsa *(*lsa-methods*)*, 29
summary.lsa_group *(*lsa-methods*)*, 29

tests, 57
tests(), *25, 26*, *28*, *30, 31*, *39*, *60*, *62*
transfer_entropy, 58
transition_probabilities, 62
transition_probabilities(), *22*, *25, 26*, *37*, *51*, *62*
transitions, 60
transitions(), *4, 5*, *9, 10*, *12*, *14*, *16, 17*, *22, 23*, *25, 26*, *28–31*, *35, 36*, *38–42*, *45*, *48*, *51, 52*, *56*, *58, 59*, *62, 63*

unregister_lsa_engine, 63
unregister_lsa_engine(), *52*