Package 'amorem'

June 29, 2026

Title Augmented Modelling of Relational Events

Version 1.0.0

Description Utilities for simulating and prototyping relational event models, including helpers to generate dynamic event sequences and covariate processes for sender and receiver sets. The endogenous-effect and case-control estimation machinery follows Juozaitiene and Wit (2024) <doi:10.1093/jrsssa/qnae132>.

License MIT + file LICENSE

Depends R (>= 4.2.0)

URL https://franciscorichter.github.io/amorem/, https://github.com/franciscorichter/amorem

Imports splines, stats, Rcpp, survival, withr

LinkingTo Rcpp

Suggests torch, testthat (>= 3.1.0), mgcv, coxme, roxygen2 (>= 7.3.0), pkgdown (>= 2.0.0), knitr, rmarkdown

Encoding UTF-8

LazyData true

Config/testthat/edition 3

RoxygenNote 7.3.3

VignetteBuilder knitr

NeedsCompilation yes

Author Francisco Richter [aut, cre],
Martina Boschi [aut],
Ernst C. Wit [aut],
Melania Lembo [aut]

Maintainer Francisco Richter <richtf@usi.ch>

Repository CRAN

Date/Publication 2026-06-29 13:50:12 UTC

Contents

`attach_static_covariates` *Attach static covariates to an event log*

Description

This helper augments an event log with sender and/or receiver covariates that live in separate lookup tables. It is designed for static covariates (one row per actor). Dynamic covariates should be merged manually before calling this helper.

Usage

```
attach_static_covariates(
  event_log,
  sender_covariates = NULL,
  receiver_covariates = NULL,
  actor_col = "actor",
  sender_prefix = "sender_",
  receiver_prefix = "receiver_",
  allow_missing = TRUE
)
```

Arguments

`event_log`	A standardized event log containing columns `sender` and `receiver`.
`sender_covariates`, `receiver_covariates`	Data frames with one row per actor. Each must include the identifier column specified by `actor_col`.
`actor_col`	Name of the identifier column inside the covariate tables.
`sender_prefix`, `receiver_prefix`	Prefixes applied to the appended covariate column names.
`allow_missing`	Logical; if `FALSE`, missing actors trigger an error.

Value

The input `event_log` with additional columns for each covariate table supplied.

classroom_actors *Classroom actor attributes (McFarland 2001)*

Description

Per-actor covariates for the classroom_events event stream.

Usage

```
classroom_actors
```

Format

A data frame with 20 rows and 3 columns:

id Character actor id matching the `sender`/`receiver` columns of classroom_events.

sex Factor `"F"` / `"M"` — biological sex.

role Factor with levels `"instructor"`, `"grade_11"`, `"grade_12"`.

Source

McFarland (2001), via `networkDynamic`. See classroom_events.

classroom_events *Classroom interaction events (McFarland 2001)*

Description

Time-stamped directed interactions among 20 individuals in a single US high-school class session, recorded on 16 October 1996 by Daniel McFarland. The same data appear in the `networkDynamic` R package as `McFarland_cls33_10_16_96`; this is a tidy event-table form.

Usage

```
classroom_events
```

Format

A data frame with 691 rows and 5 columns:

time Minutes since the start of the class period.

sender Character actor id matching classroom_actors`$id`.

receiver Character actor id matching classroom_actors`$id`.

interaction_type Factor with levels `"social"`, `"sanction"`, `"task"`.

weight Integer weight of the interaction.

Source

McFarland, D. (2001). Student resistance: How the formal and informal organization of classrooms facilitate everyday forms of student defiance. *American Journal of Sociology* 107(3), 612–678. doi:10.1086/338779. Redistributed via the `networkDynamic` R package (CRAN), dataset `McFarland_cls33_10_16_96`.

See Also

classroom_actors

`college_msg` *CollegeMsg: private messages on a university online community*

Description

Directed time-stamped instant messages between students of the University of California, Irvine over 193 days in 2004. Each row is one message. Sourced from the SNAP repository.

Usage

```
college_msg
```

Format

A data frame with 59,835 rows and 3 columns:

time Days since the first message. `attr(., "unix_origin")` holds the Unix epoch of `time = 0`.

sender Character user id.

receiver Character user id.

Source

Panzarasa, P., Opsahl, T., Carley, K. (2009). Patterns and dynamics of users' behavior and interaction: Network analysis of an online community. *Journal of the American Society for Information Science and Technology* 60(5), 911–932. Distributed via SNAP: `https://snap.stanford.edu/data/CollegeMsg.html`.

`compare_models` *Compare candidate endogenous specifications by AIC*

Description

Superseded by `rem()`, the unified front-end for fitting relational event models on preprocessed case-control data. `compare_models()` remains fully supported.

Convenience wrapper that runs the canonical case-control / no-intercept binomial-GLM recipe on every specification in `models` and returns a tidy AIC comparison table. One case-control sample is drawn from `event_log` and shared across every specification so that the AIC values are directly comparable.

Each specification is a character vector of stat names accepted by `endogenous_features()`. The function computes the union of all stats once, builds case-minus-control differences, and fits one binomial GLM per specification with the appropriate subset of columns. The fitted models are equivalent to the partial-likelihood parametrisation used in case-control REM inference (Vu et al. 2017; Juozaitienė & Wit 2024).

For `n_controls = 1` the helper fits a no-intercept binomial GLM on case-minus-control differences. For `n_controls > 1` it falls back to `survival::clogit()` — a true conditional-logistic fit that correctly handles multiple controls per stratum. The `survival` package is in Suggests and is required only when `n_controls > 1`.

Usage

```
compare_models(
  event_log,
  models,
  n_controls = 1,
  scope = c("all", "appearance", "citation"),
  mode = c("one", "two"),
  random_effects = NULL,
  half_life = NULL,
  seed = NULL,
  keep_fits = FALSE
)
```

Arguments

`event_log`	Data frame with `sender`, `receiver`, and `time` columns.
`models`	Named list of character vectors. Each entry names one candidate specification; the vector contents are the endogenous statistics it includes. Stats must be valid names for `endogenous_features()`.
`n_controls`	Number of controls per case in `sample_non_events()`. `1` uses a binomial GLM on differences; `> 1` uses `survival::clogit()` on the stratified case-control table.
`scope, mode`	Passed through to `sample_non_events()`; see that help page for semantics.

`random_effects` Optional character vector. May be any of `NULL` (no random effects), `"sender"`, `"receiver"`, or `c("sender", "receiver")`. When supplied, requires `n_controls > 1`. With one axis, the stratified coxph fit adds a Gamma `survival::frailty()` term on the requested actor. With both axes, the fit dispatches to `coxme::coxme()` with two normal `~ 1 | actor` random effects, which avoids `survival::coxph`'s one-sparse-term cap and the dense-penalty segfault on stratified case-control designs. The two-axis path requires the `coxme` package (Suggests). This is the actor-heterogeneity correction used by Juozaitienė & Wit (2024) and changes which specification AIC selects on real-world data (timing variants typically win over count baselines once actor effects are absorbed). Defaults to `NULL` (no random effects).

`half_life` Required when any specification contains an exp-decay stat. Shared across all specs that use one.

`seed` Optional integer seed for the case-control sample.

`keep_fits` Logical; when `TRUE`, the returned table carries the fitted model objects (one per spec, named by model, `NULL` for specs that failed) as `attr(result, "fits")`, e.g. for plotting estimated effects. Defaults to `FALSE`.

Value

A data frame with one row per specification and columns `model`, `n_terms`, `n_obs`, `log_lik`, `AIC`, `delta_AIC`. Sorted ascending by `AIC`. The model with the lowest AIC has `delta_AIC = 0`.

References

Juozaitienė R, Wit EC (2024). It's about time: revisiting reciprocity and triadicity in relational event analysis. *Journal of the Royal Statistical Society Series A* 188(4), 1246-1262. doi:10.1093/jrsssa/qnae132.

See Also

`endogenous_features()`, `sample_non_events()`.

Examples

```
data(classroom_events)
compare_models(
  classroom_events,
  models = list(
    count       = c("reciprocity_count", "transitivity_count"),
    continuous  = c("reciprocity_time_recent",
                    "transitivity_time_recent"),
    interrupted = c("reciprocity_time_recent_interrupted",
                    "transitivity_time_recent_interrupted")),
  seed = 11)
```

`compare_models_global` *Compare REM specifications with global covariate effects*

Description

Superseded by `rem()`, the unified front-end for fitting relational event models on preprocessed case-control data. `compare_models_global()` remains fully supported.

Implements the time-shifted partial likelihood of Lembo, Juozaitienė, Vinciotti & Wit (2025) for fitting relational event models with **global covariate effects** — covariates that are time-dependent but constant across all interacting pairs (e.g.\ temperature, time of day, the residual baseline hazard). Standard case-control partial likelihood cannot identify these because global terms cancel in the rate ratio; this function follows the paper's Section 4 recipe: a random per-dyad time shift breaks the cancellation, and with one non-event per event the partial likelihood reduces to a degenerate logistic additive model fit by `mgcv::gam()`.

Per the paper's equations 11-13:

$$\mathcal{L}^{PS}(f,g) = \prod_{k=1}^{n} \frac{\exp\{\Delta_k(f; x_{s_k r_k}) + \Delta_k(g; x_k)\}}{1 + \exp\{\Delta_k(f; x_{s_k r_k}) + \Delta_k(g; x_k)\}}$$

where each Δ_k is the difference between the (smooth) function evaluated at the focal event time and at the sampled non-event's *shifted* time $t_k^* = t_k - h_{s^* r^*}$.

Shift distribution. Per-dyad shifts H_{sr} are drawn independently from an exponential distribution with mean $\nu \cdot \bar{\Delta} t$ where $\bar{\Delta} t$ is the average inter-arrival time in `event_log`. The paper's simulation studies find that $\nu = 1$ works in practice and that the estimates are robust to choices in $[0.1, 10]$.

Specification format. Each entry of `models` is a named character vector mapping a covariate name (a statistic in `endogenous_features()` **or** a column of `global_covariates`) to an effect type:

- `"linear"` – linear `beta * x` term.
- `"nl"` – smooth `s(x)` (thin-plate, paper's default).
- `"tv"` – smooth `s(time, by = x)` (time-varying).
- `"tvnl"` – tensor product `te(time, x)`.
- `"global_smooth"` – smooth `s(x_global)` evaluated at the focal time vs. the non-event's shifted time (the paper's `g_b(x^{(b)}(t))` family).
- `"global_cyclic"` – cyclic smooth `s(x_global, bs = "cc")` for time-of-day-like covariates with a periodic domain.
- `"global_time"` – a smooth on `time` itself, recovering the residual time effect $g_0(t)$ of paper eq. 3.

Usage

```
compare_models_global(
  event_log,
  models,
  global_covariates = NULL,
```

```
  scope = c("all", "appearance", "citation"),
  mode = c("one", "two"),
  half_life = NULL,
  shift_scale = 1,
  k = NULL,
  k_cyclic = 10,
  seed = NULL,
  keep_fits = FALSE
)
```

Arguments

event_log	Data frame with sender, receiver, time.
models	Named list of specifications (see "Specification format" above).
global_covariates	Optional data frame with a time column plus one column per global covariate referenced in models. The function evaluates each covariate at the focal event time and at the non-event's shifted time by stepwise lookup (LOCF on the time axis).
scope, mode	Passed through to sample_non_events().
half_life	Required when any dyadic spec uses an exp-decay stat.
shift_scale	Multiplier on the average inter-arrival time for the exponential shift distribution. Defaults to 1.
k	Optional knot count for smooth terms (see mgcv::s()). Defaults to mgcv's automatic choice.
k_cyclic	Knot count for global_cyclic smooths (paper uses 10 for time-of-day).
seed	Integer seed for the case-control sample and the shift draws.
keep_fits	Logical; when TRUE, the returned table carries the fitted model objects (one per spec, named by model, NULL for specs that failed) as attr(result, "fits"), e.g. for plotting estimated effects. Defaults to FALSE.

Value

Data frame with one row per specification and columns model, n_terms, n_obs, log_lik, AIC, delta_AIC.

References

Lembo M, Juozaitienė R, Vinciotti V, Wit EC (2025). *Relational event models with global covariates: an application to bike sharing*. Journal of the Royal Statistical Society, Series C. doi:10.1093/jrsssc/qlaf058.

See Also

compare_models() (linear, no globals), compare_models_smooth() (smooth dyadic effects, no globals).

Examples

```
data(classroom_events)
# Hourly temperature track on the same time axis:
g <- data.frame(time = seq(0, max(classroom_events$time), length = 50),
                temperature = rnorm(50, 20, 5))
compare_models_global(
  classroom_events,
  models = list(
    dyadic_only = c(reciprocity_count        = "linear",
                    transitivity_count       = "linear"),
    with_global = c(reciprocity_count        = "linear",
                    transitivity_count       = "linear",
                    temperature              = "global_smooth",
                    time                     = "global_time")),
  global_covariates = g,
  seed = 11, k = 5)
```

`compare_models_smooth` *Compare candidate specifications with smooth (TV / NL / TVNL) effects*

Description

Superseded by `rem()`, which fits the same smooth (TV / NL / TVNL) effects on preprocessed case-control data. `compare_models_smooth()` remains fully supported.

Mirrors `compare_models()` but lets each statistic in a specification take one of four effect types instead of a single linear coefficient: linear, time-varying (TV), non-linear (NL), or jointly time-varying non-linear (TVNL). The smooth machinery follows Boschi, Lerner & Wit (2025); the matrix-of-event-vs-non-event trick is documented in their Section 3.3.

For each specification:

- One case-control sample is drawn from `event_log` with `n_controls = 1` (paired event / non-event design).
- For every requested statistic, both the case (event) and the control (non-event) features are computed via `endogenous_features()`.
- The mgcv design uses the case-vs-control matrix trick:
  - linear -> a single coefficient on `case - control` (column `d_stat`).
  - tv -> `s(time, by = d_stat)` — smooth in time, multiplied by `d_stat`.
  - nl -> `s(stat_mat, by = I_mat)` where `stat_mat` is a two-column matrix `cbind(case, control)` and `I_mat` is `cbind(1, -1)`.
  - tvnl -> `te(time_mat, stat_mat, by = I_mat)` tensor product smooth, with time_mat both columns equal to the event time vector.
- The model is fitted with `mgcv::gam` and a degenerate logistic likelihood: response = `rep(1, n)`, formula = `one ~ -1 + ...`, `family = binomial`. This matches Boschi et al. equation 8.

AIC values are directly comparable across specifications because every fit uses the same case-control sample. Returns the same tidy `data.frame` as `compare_models()`.

Usage

```
compare_models_smooth(
  event_log,
  models,
  scope = c("all", "appearance", "citation"),
  mode = c("one", "two"),
  half_life = NULL,
  k = NULL,
  seed = NULL,
  keep_fits = FALSE
)
```

Arguments

event_log	Data frame with sender, receiver, time columns.
models	Named list of specifications. Each entry is itself a named character vector (or named list) mapping statistic names to effect types: "linear", "tv", "nl", or "tvnl". Example:

```
list(
  linear = c(reciprocity_count   = "linear",
             transitivity_count  = "linear"),
  nl     = c(reciprocity_time_recent  = "nl",
             transitivity_time_recent = "nl"),
  tvnl   = c(reciprocity_time_recent  = "tvnl",
             transitivity_time_recent = "tvnl"))
```

scope, mode	Passed through to sample_non_events().
half_life	Required when an exp-decay statistic is requested.
k	Optional integer: knot count for s() and te() terms. Default NULL lets mgcv choose (-1).
seed	Integer seed for the case-control sample.
keep_fits	Logical; when TRUE, the returned table carries the fitted model objects (one per spec, named by model, NULL for specs that failed) as attr(result, "fits"), e.g. for plotting estimated effects. Defaults to FALSE.

Value

Data frame with one row per specification and columns model, n_terms, n_obs, log_lik, AIC, delta_AIC.

References

Boschi M, Lerner J, Wit EC (2025). *Beyond Linearity and Time-Homogeneity: Relational Hyper Event Models with Time-Varying Non-Linear Effects*. arXiv:2509.05289.

See Also

compare_models() for the linear-only variant.

Examples

```
data(classroom_events)
compare_models_smooth(
  classroom_events,
  models = list(
    linear = c(reciprocity_time_recent  = "linear",
               transitivity_time_recent = "linear"),
    nl     = c(reciprocity_time_recent  = "nl",
               transitivity_time_recent = "nl"),
    tvnl   = c(reciprocity_time_recent  = "tvnl",
               transitivity_time_recent = "tvnl")),
  seed = 11)
```

---

cpp_supported_stats *Endogenous statistics with a compiled fast path*

Description

Returns the names of the endogenous statistics that endogenous_features() evaluates with the compiled C++ engine. Statistics outside this set are computed by the (slower) pure-R fallback.

Value

A character vector of statistic names.

See Also

endogenous_features()

Examples

```
length(cpp_supported_stats())
head(cpp_supported_stats())
```

dist_matrix *US state distance matrix*

Description

A 56 × 56 matrix of pairwise geographic distances (in metres) between US states and territories, computed from boundary geometries via `sf::st_distance`.

Usage

```
dist_matrix
```

Format

A numeric matrix with 56 rows and 56 columns. Row and column names are state/territory names.

Source

Computed from US Census TIGER/Line shapefiles using the **tigris**, **sf**, and **geosphere** packages. See Walker (2024), Pebesma (2018), Hijmans (2022).

email_eu_core *Email-Eu-Core temporal (single-department subset)*

Description

Internal emails between members of a single department of a large European research institution over ~803 days. The dataset has been filtered to remove self-loops. Sourced from the SNAP repository as a single-department slice of the email-Eu-core-temporal dataset.

Usage

```
email_eu_core
```

Format

A data frame with 12,216 rows and 3 columns:

time Days since the first email in the recording window.

sender Character employee id (anonymised).

receiver Character employee id (anonymised).

Source

Paranjape, A., Benson, A.R., Leskovec, J. (2017). Motifs in temporal networks. *WSDM '17*, 601–610. doi:10.1145/3018661.3018731. Distributed via SNAP: https://snap.stanford.edu/data/email-Eu-core-temporal.html.

endogenous_features *Compute endogenous event-network statistics*

Description

Given a standardized relational event log, this helper derives historical statistics for each event based on the evolving network. The statistics follow the taxonomy of Juozaitienė and Wit (2025, JRSS-A) and cover reciprocity, transitivity, cyclic closure, sending balance and receiving balance. All definitions use the *continuous* convention (effects persist even after a closure event).

Usage

```
endogenous_features(
  event_log,
  stats = c("sender_outdegree", "receiver_indegree", "reciprocity", "recency"),
  half_life = NULL,
  sort = TRUE,
  history_log = NULL,
  prior_log = NULL
)
```

Arguments

event_log	A data.frame containing at least sender, receiver, and time columns.
stats	Character vector of statistics to compute. See **Details** for the full list of allowed values.
half_life	Positive numeric; the half-life parameter T for exponential-decay statistics (*_exp_decay*).
sort	Logical; when TRUE, events are ordered by time prior to computing summaries (ties preserve input order).
history_log	Optional data.frame giving the authoritative event history (columns sender, receiver, time). When supplied, only rows of event_log whose (sender, receiver, time) triple appears in history_log update the running network state; all other rows (e.g. sampled non-events / controls) have their statistics computed against that history but never enter it. This makes it possible to evaluate endogenous statistics for non-events without those non-events polluting the history. Defaults to NULL (every row is treated as an event). Currently supported only for statistics handled by the C++ engine (see cpp_supported_stats()).
prior_log	Optional data.frame of events that precede the study window (columns sender, receiver, time), used to **warm-start** the network state. Its rows always update the running state but never appear in the returned data.frame. This separates warm-starting from the non-event masking role of history_log: pass earlier history through prior_log and use history_log purely to mark which rows of event_log are real events. Defaults to NULL. Like history_log, it is currently supported only for statistics handled by the C++ engine (see cpp_supported_stats()).

Details

All statistics are evaluated immediately **before** the event is logged. They are grouped into five families.

Degree / baseline:

`sender_outdegree` Number of events previously sent by the sender.

`receiver_indegree` Number of events previously received by the receiver.

`recency` Elapsed time since the last event on the same ordered pair; `NA` when the dyad is brand new.

Reciprocity — reverse-dyad (receiver $\to$ sender) history:

`reciprocity` / `reciprocity_binary` 1 if the reverse dyad has ever been observed, 0 otherwise.

`reciprocity_count` Total count of past reverse-dyad events.

`reciprocity_exp_decay` Exponentially weighted sum of past reverse-dyad events (requires `half_life`).

`reciprocity_time_recent` Elapsed time since the most recent reverse-dyad event; `NA` if none.

`reciprocity_time_first` Elapsed time since the first reverse-dyad event; `NA` if none.

Transitivity — two-path $s \to k \to r$:

`transitivity_binary` 1 if any intermediary k exists with both (s, k) and (k, r) before t.

`transitivity_count` Number of such intermediaries.

`transitivity_binary_ordered` Like binary but requiring (s, k) to precede (k, r).

`transitivity_count_ordered` Count with order restriction.

`transitivity_exp_decay` Exp-decay weighted sum over two-paths (requires `half_life`).

`transitivity_exp_decay_ordered` Exp-decay with order restriction.

`transitivity_time_recent` Time since the most recently completed two-path; `NA` if none.

`transitivity_time_first` Time since the earliest two-path; `NA` if none.

`transitivity_time_recent_ordered` Time since the most recent ordered two-path; `NA` if none.

`transitivity_time_first_ordered` Time since the earliest ordered two-path; `NA` if none.

Cyclic closure — two-path $r \to k \to s$, closed by $s \to r$:

`cyclic_binary` 1 if any cyclic two-path exists.

`cyclic_count` Number of cyclic intermediaries.

`cyclic_time_recent` Time since the most recent cyclic two-path formation; `NA` if none.

`cyclic_time_first` Time since the first cyclic two-path formation; `NA` if none.

Sending balance — shared target: both $s \to k$ and $r \to k$ exist:

`sending_balance_binary` 1 if any shared target exists.

`sending_balance_count` Number of shared targets.

`sending_balance_time_recent` Time since the most recent shared-target two-path formation; `NA` if none.

`sending_balance_time_first` Time since the first shared-target two-path formation; `NA` if none.

Receiving balance — shared source: both $k \to s$ and $k \to r$ exist:

`receiving_balance_binary` 1 if any shared source exists.

`receiving_balance_count` Number of shared sources.

`receiving_balance_time_recent` Time since the most recent shared-source two-path formation; `NA` if none.

`receiving_balance_time_first` Time since the first shared-source two-path formation; `NA` if none.

The statistic `"sender_receivers_set"` is special: it adds a **list-column** in which each element is the character vector of receivers the row's sender has reached before that row (the building block for set-valued endogenous covariates, e.g. an alien species' previously invaded regions). It honours `history_log`, so it can be computed for sampled non-events without those non-events polluting the history.

Value

The event log with added columns, one per requested statistic (`sender_receivers_set` is added as a list-column).

Examples

```
data(classroom_events)
feats <- endogenous_features(classroom_events,
                                     stats = c("reciprocity", "recency"))
head(feats)
```

`gof_auxiliary` *GOF test for an auxiliary (unmodelled) statistic*

Description

Implements the auxiliary-statistic test of Boschi & Wit (2025), Section 3.7 / eq. 20. Tests whether a covariate `auxiliary` that is *not* part of `model` has nonetheless been adequately captured indirectly by the fitted model. Uses the simulation-based p-value described in the paper: `n_sim` replicates of $G^*[\hat{\gamma}, u]$ are drawn from i.i.d. standard normals, the test statistic $T_\phi = \sup_u |G[\hat{\gamma}, u]|$ is computed, and the empirical p-value is the fraction of replicates with $T^*_{\phi,b} \geq T_\phi$.

Usage

```
gof_auxiliary(
  event_log,
  model,
  auxiliary,
  n_sim = 1000,
  scope = "all",
```

```
  mode = "one",
  half_life = NULL,
  seed = NULL
)
```

Arguments

`event_log`	Dyadic event log.
`model`	Named character vector of `<stat> = "linear"` mapping for the *fitted* covariates (must not contain `auxiliary`).
`auxiliary`	Name of the statistic to test as an unmodelled feature; must be a statistic computable by `endogenous_features()`.
`n_sim`	Number of Monte Carlo replicates (default 1000).
`scope`, `mode`, `half_life`, `seed`	See `compare_models()`.

Value

List with `statistic` (T_ϕ), `p_value`, `G`, `u`, and `auxiliary`.

References

Boschi M, Wit EC (2025). *Goodness of fit in relational event models*. Statistics and Computing 36(4).

`gof_global` *Omnibus GOF test via Cauchy combination*

Description

Implements the omnibus test of Boschi & Wit (2025), Section 3.6 / eq. 19. Runs `gof_univariate()` per covariate in `model`, then combines the resulting p-values via the Cauchy combination $T_o = \frac{1}{L}\sum_l \tan(\pi(0.5 - P_l))$ (Liu & Xie 2020), with analytic p-value $\frac{1}{2} - \arctan(T_o)/\pi$.

Usage

```
gof_global(
  event_log,
  model,
  scope = "all",
  mode = "one",
  half_life = NULL,
  seed = NULL
)
```

Arguments

event_log	Dyadic event log.
model	Named character vector of <stat> = "linear" mapping.
scope, mode, half_life, seed	See compare_models().

Value

List with statistic (T_o), p_value, and components (per-covariate data.frame with covariate, statistic, p_value).

References

Boschi M, Wit EC (2025). *Goodness of fit in relational event models*. Statistics and Computing 36(4).

---

gof_multivariate *Multivariate GOF test for smooth or random-effect covariates*

Description

Implements the multivariate test of Boschi & Wit (2025), Section 3.4. Builds a q-dimensional cumulative residual process from the spline basis of the requested covariate's smooth effect, normalises by the inverse-square-root of the empirical variance-covariance matrix $\hat{J}$ (eq. 17), and tests against a q-dimensional standard Brownian bridge via $T_\psi = \sup_u \|\hat{W}\|^2$. The p-value is computed empirically by simulating n_sim Brownian bridge trajectories.

Usage

```
gof_multivariate(
  event_log,
  model,
  covariate,
  k_basis = 5,
  n_sim = 1000,
  scope = "all",
  mode = "one",
  half_life = NULL,
  seed = NULL
)
```

Arguments

event_log	Dyadic event log.
model	Named character vector of <stat> = "linear" mapping (for the rest of the model); the test target is covariate with a flexible smooth basis of dimension k_basis - 1.

`covariate`	Name of the covariate to test under a smooth effect.
`k_basis`	Spline-basis dimension for `covariate` (passed as `k` to `mgcv::s()`; the resulting design matrix has `k_basis - 1` columns under thin-plate identifiability constraints).
`n_sim`	Number of simulated Brownian bridges for the empirical p-value (default 1000).
`scope, mode, half_life, seed`	See `compare_models()`.

Value

List with `statistic` (T_ψ), `p_value`, `W` (n x q matrix), `u`, and `covariate`.

References

Boschi M, Wit EC (2025). *Goodness of fit in relational event models*. Statistics and Computing 36(4).

`gof_univariate` *Goodness-of-fit test for a single FLE covariate*

Description

Implements the univariate cumulative martingale residual test of Boschi & Wit (2025), Section 3.3. The test statistic is $T_x = \sup_u |\hat{W}[u]|$ where $\hat{W}[u]$ is the normalised cumulative score process for the requested covariate; under correct specification $\hat{W}$ converges to a standard Brownian bridge, so the p-value follows the Kolmogorov-Smirnov distribution $2\sum_{k\geq 1}(-1)^{k-1}e^{-2k^2t^2}$.

Usage

```
gof_univariate(
  event_log,
  model,
  covariate,
  scope = "all",
  mode = "one",
  half_life = NULL,
  seed = NULL
)
```

Arguments

`event_log`	Dyadic event log.
`model`	Named character vector of `<stat> = "linear"` mapping.
`covariate`	Name of the covariate in `model` to test.
`scope, mode, half_life, seed`	See `compare_models()`.

Value

A list with `statistic` (T_x), `p_value` (KS), `W` (numeric vector of length `n`, the normalised process), and `u` (the time grid in `[0, 1]`).

References

Boschi M, Wit EC (2025). *Goodness of fit in relational event models*. Statistics and Computing 36(4).

Examples

```
data(classroom_events)
gof_univariate(classroom_events,
  model = c(reciprocity_count  = "linear",
            transitivity_count = "linear"),
  covariate = "reciprocity_count", seed = 1)
```

---

hyperedge_activity *Activity counter for hyperedge subsets*

Description

For a focal candidate hyperedge (t, I, J), `activity(t, I, J)` counts the number of past events (t_m, I_m, J_m) with $t_m < t$ satisfying $I \subseteq I_m$ AND $J \subseteq J_m$.

Usage

```
hyperedge_activity(hyperedge_log, I, J = character(0), t)
```

Arguments

`hyperedge_log`	A hyperedge log (see `hyperedge_log()`).
`I`	Character vector of sender names defining the focal subset.
`J`	Character vector of receiver names defining the focal subset. Pass `character(0)` to ignore the receiver side (undirected events).
`t`	Focal time. Only events strictly before `t` contribute.

Value

A single non-negative integer.

References

Lerner J, Boschi M, Wit EC (2025). Subset repetition.

`hyperedge_features` *Endogenous features for a hyperedge event log*

Description

Hyperedge analogue of `endogenous_features()`. Accepts a hyperedge log (see `hyperedge_log()`) and computes hyperedge-native statistics, falling back to the dyadic engine for stat names that belong to the standard dyadic endogenous catalogue.

Usage

```
hyperedge_features(hyperedge_log, stats, half_life = NULL)
```

Arguments

`hyperedge_log`	A hyperedge log (see `hyperedge_log()`).
`stats`	Character vector of statistic names. Mix of hyperedge- native names listed above and the dyadic catalogue names accepted by `endogenous_features()`.
`half_life`	Required when an exp-decay statistic is requested (only applies to delegated dyadic stats; hyperedge subrep does not use a half-life).

Details

Recognised hyperedge stat names:

`"subrep_<rho>_<l>"` Directed subset repetition (paper eq. 4). `rho` = sender-side subset cardinality (1..|I|), `l` = receiver-side subset cardinality (0..|J|, 0 = ignore receivers). Examples: `"subrep_1_1"` (average activity over single-actor sub-pairs), `"subrep_2_1"` (over pair-of-senders × single-receiver subpairs).

`"subrep_<rho>"` Undirected subset repetition. Equivalent to `"subrep_<rho>_0"`; counts past events whose participant set is a superset of the chosen subset, with no receiver-side restriction.

`"activity"` Counts past events whose participant set covers the focal event's entire `(I, J)` pair. Equivalent to `"subrep_<|I|>_<|J|>"`.

For dyadic-shaped events (every row has `|I| = |J| = 1`) and a dyadic stat name, this function delegates to `endogenous_features()` via `as_dyadic_log()`.

Value

The hyperedge log with one added column per requested stat.

References

Boschi M, Lerner J, Wit EC (2025). *Beyond Linearity and Time-Homogeneity: Relational Hyper Event Models with Time-Varying Non-Linear Effects*. arXiv:2509.05289.

See Also

hyperedge_subrep(), hyperedge_activity(), endogenous_features().

Examples

```
hl <- hyperedge_log(
  I    = list(c("a","b"), c("a","c"), c("b","c"), c("a","b","c")),
  J    = list(c("X"),     c("X","Y"), c("Y"),     c("X")),
  time = c(1, 2, 3, 4))
hyperedge_features(hl,
  stats = c("subrep_1_1", "subrep_2_1", "activity"))
```

---

hyperedge_log *Build / detect / convert hyperedge event logs*

Description

A *hyperedge log* generalises the dyadic (sender, receiver, time) event log used elsewhere in amorem to a (I, J, time) event log where I and J are list-columns containing the set of senders and the set of receivers participating in each hyperevent. This matches the data model of Boschi, Lerner & Wit (2025): each event is a time-stamped directed hyperedge (t_m, I_m, J_m) from a sender set to a receiver set.

Usage

```
hyperedge_log(I, J, time)

is_hyperedge_log(x)

as_hyperedge_log(event_log)

as_dyadic_log(hyperedge_log)
```

Arguments

I	List-column: each element is a character vector of sender actor names participating in that event. Length-1 vectors are allowed (and become standard dyadic events when combined with a length-1 J).
J	List-column: each element is a character vector of receiver actor names. Empty character vectors are allowed and signal an *undirected* hyperevent.
time	Numeric vector of event times. Must be finite and non-decreasing after sorting.
x	A data frame or list-of-columns to test or convert.
event_log	A dyadic event log with sender, receiver, time columns.
hyperedge_log	A hyperedge log produced by hyperedge_log() or as_hyperedge_log().

Details

The constructor `hyperedge_log()` accepts list-columns directly and performs validation (character members, non-empty sets, finite times, sorted by time). `as_hyperedge_log()` promotes a dyadic `(sender, receiver, time)` data frame to the hyperedge form by wrapping each `sender` and `receiver` in a length-1 character vector. `as_dyadic_log()` is the inverse: it succeeds only when every row of the hyperedge log has a length-1 sender set AND a length-1 receiver set.

For *undirected* hyperevents (e.g.\ multi-actor meetings), pass an empty receiver set: `J = list(character(0), character(0), ...)`. The receiver list-column must still be present.

Value

A `data.frame` with columns `I`, `J`, `time`, additionally carrying class `amorem_hyperedge_log` to distinguish it from a dyadic log in dispatch contexts. Sorted by `time` ascending.

References

Boschi M, Lerner J, Wit EC (2025). *Beyond Linearity and Time-Homogeneity: Relational Hyper Event Models with Time-Varying Non-Linear Effects*. arXiv:2509.05289.

Examples

```
# Two co-authored citation events:
hl <- hyperedge_log(
  I    = list(c("alice", "bob"), c("alice", "carol")),
  J    = list(c("paperA"), c("paperA", "paperB")),
  time = c(1.0, 2.5))
is_hyperedge_log(hl)

# Round-trip a dyadic log:
dy <- data.frame(sender = c("a", "b"),
                 receiver = c("b", "c"),
                 time = c(1, 2))
h <- as_hyperedge_log(dy)
as_dyadic_log(h)
```

---

hyperedge_sizes *Cardinality columns for a hyperedge event log*

Description

Adds two integer columns to a hyperedge log: `size_I` (the number of senders) and `size_J` (the number of receivers). Convenient shortcut for filtering / case-control sampling matched on cardinality (see Boschi et al. 2025, Section 3.3).

Usage

```
hyperedge_sizes(hyperedge_log)
```

Arguments

`hyperedge_log`	A hyperedge log.

Value

The hyperedge log with two added integer columns.

`hyperedge_subrep` *Subset repetition statistic for a hyperedge event log*

Description

For a focal hyperedge (t, I, J) and orders (ρ, ℓ), computes the **average activity** over every sender subset of `I` of size `rho` and every receiver subset of `J` of size `l`, per Boschi, Lerner & Wit (2025) Equation 4:

$$\text{subrep}^{\rho,\ell}(t, I, J) = \frac{1}{\binom{|I|}{\rho}\binom{|J|}{\ell}} \sum_{I' \subseteq I,\ |I'|=\rho} \sum_{J' \subseteq J,\ |J'|=\ell} \text{activity}(t, I', J').$$

Usage

```
hyperedge_subrep(
  hyperedge_log,
  I,
  J = character(0),
  t,
  rho = length(I),
  l = length(J)
)
```

Arguments

`hyperedge_log`	A hyperedge log (see `hyperedge_log()`).
`I`	Character vector of senders for the focal event.
`J`	Character vector of receivers (or `character(0)` for undirected).
`t`	Focal time.
`rho`	Order on the sender side: subset cardinality. Must be between 1 and `length(I)`. Defaults to `length(I)` (full subset).
`l`	Order on the receiver side: subset cardinality. Must be between 0 and `length(J)`. Defaults to `length(J)` (full subset); pass 0 to ignore receivers (undirected).

Details

For dyadic events with $|I| = |J| = 1$, `subrep(rho = 1, l = 1)` reduces to the dyad event count (already exposed as `reciprocity_count` and related stats in `endogenous_features()`). The function exists because for true hyperedge data the average over subsets of intermediate size captures partial-subset repetition that no dyadic statistic can represent.

Value

A single non-negative numeric.

References

Boschi M, Lerner J, Wit EC (2025). *Beyond Linearity and Time- Homogeneity: Relational Hyper Event Models with Time-Varying Non-Linear Effects*. arXiv:2509.05289. Lerner J, et al. (2025). The eventnet computation framework.

Examples

```
hl <- hyperedge_log(
  I    = list(c("a","b"), c("a","c"), c("b","c")),
  J    = list(c("X"),     c("X","Y"), c("Y")),
  time = c(1, 2, 3))
# Activity for the (a, X) sub-pair before t = 4:
hyperedge_activity(hl, I = "a", J = "X", t = 4)
# First-order subrep on event (a, b) -> X at t = 4:
hyperedge_subrep(hl, I = c("a","b"), J = "X", t = 4, rho = 1, l = 1)
```

---

martingale_residuals *Martingale residuals from a case-control partial-likelihood fit*

Description

Computes per-observation martingale residuals $M_i = y_i - \pi_i$ from a one-control-per-case partial-likelihood fit, where y_i is the case indicator inside the (case, control) pair and

$$\pi_i \;=\; \frac{\exp(\eta_i)}{\exp(\eta_{\text{case}}) + \exp(\eta_{\text{ctrl}})}$$

is the fitted probability that observation i is the event in its risk set. The residuals sum to zero within each stratum.

Usage

```
martingale_residuals(
  event_log,
  model,
  scope = c("all", "appearance", "citation"),
  mode = c("one", "two"),
  half_life = NULL,
  seed = NULL
)
```

Arguments

event_log	Dyadic event log (see standardize_event_log()).
model	A named character vector mapping statistic name to "linear". Mirrors a single entry of compare_models()'s models argument. Non-linear effect types are currently rejected.
scope, mode, half_life, seed	Same meaning as in compare_models(); control the case-control sampling and the feature computation.

Details

Useful as a goodness-of-fit diagnostic: plotting residuals vs. time or vs. a covariate reveals systematic miscalibration. The convention matches survival::residuals.coxph(type = "martingale") for the two-element risk set induced by 1-control case-control sampling.

Only the linear partial-likelihood path (compare_models()-style linear-effect specs) is supported by this helper; for smooth-effect fits the case-vs-control matrix design used by compare_models_smooth() does not have a clean per-observation martingale interpretation.

Value

A data frame with one row per observation in the case-control table (so 2N rows for N events), with columns: stratum, role ("case" or "control"), sender, receiver, time, eta, fitted_prob, residual.

References

Therneau TM, Grambsch PM, Fleming TR (1990). *Martingale-based residuals for survival models*. Biometrika 77(1), 147–160.

See Also

compare_models(), compare_models_smooth().

Examples

```
data(classroom_events)
res <- martingale_residuals(
  classroom_events,
  model = c(reciprocity_count = "linear",
            transitivity_count = "linear"),
  seed = 1)
plot(res$time, res$residual,
     col = ifelse(res$role == "case", "red", "grey60"),
     ylab = "Martingale residual", xlab = "Event time")
abline(h = 0)
```

nn_control *Control parameters for the neural-network backend of rem()*

Description

Collects the architecture and training hyper-parameters used by `rem(method = "nn")`. Training maximizes the same conditional-logistic partial likelihood as `method = "clogit"` (softmax over each risk set), so this backend is a drop-in flexible counterpart of the linear conditional logit. Two predictor architectures are available:

`"mlp"` a multilayer perceptron scoring the full covariate vector jointly — can represent interactions between statistics.

`"additive_spline"` an additive predictor `sum_k f_k(x_k)` with each `f_k` a B-spline expansion fitted by (mini-batch) stochastic gradient — the STREAM construction of Filippi-Mazzola & Wit (2024, JRSS-C 73(4), doi:10.1093/jrsssc/qlae023). Interpretable per-feature curves; with `batch_strata` it scales to event logs far beyond what an in-memory smooth fit can hold.

Usage

```
nn_control(
  hidden = c(16L, 8L),
  activation = c("relu", "tanh"),
  architecture = c("mlp", "additive_spline"),
  spline_df = 8L,
  batch_strata = NULL,
  epochs = 300L,
  lr = 0.01,
  l2 = 1e-04,
  validation = 0.2,
  patience = 25L,
  standardize = TRUE,
  engine = c("r", "torch"),
  seed = NULL,
  verbose = FALSE
)
```

Arguments

`hidden`	Integer vector of hidden-layer sizes for `"mlp"`, e.g. `c(16, 8)`. Use `integer(0)` for no hidden layer (recovers a linear conditional logit fit by gradient descent). Ignored for `"additive_spline"`.
`activation`	Hidden-layer activation for `"mlp"`: `"relu"` or `"tanh"`.
`architecture`	Predictor architecture: `"mlp"` (default) or `"additive_spline"`; see *Description*.
`spline_df`	Degrees of freedom (basis size) per covariate for `"additive_spline"`; passed to `splines::bs()`.

batch_strata	Optional mini-batch size, in **strata**, for stochastic gradient training. NULL (default) trains full-batch; a value such as 512 takes one Adam step per sampled chunk of strata each epoch.
epochs	Maximum number of training epochs (full passes over the training strata).
lr	Adam learning rate.
l2	L2 penalty (weight decay). The pure-R engine penalises the weights only; the torch engine applies it via Adam's weight_decay.
validation	Fraction of strata held out for validation / early stopping. Set to 0 to train on everything (no early stopping).
patience	Early-stopping patience: training stops after this many epochs without improvement of the validation loss; the best parameters are restored.
standardize	Z-score the features before training (recommended; the scaling is stored and re-applied by predict()).
engine	Training engine: "r" (default) uses the built-in pure-R implementation with hand-derived gradients; "torch" trains the *same* model and loss with the **torch** package (libtorch / autograd), which is markedly faster and, with batch_strata, scales to large event logs (optionally on GPU). The two engines fit identical model classes and return interchangeable objects. "torch" requires the suggested **torch** package (run torch::install_torch() once) and equal-sized strata (the usual case-control layout with a fixed number of controls).
seed	Optional integer seed for reproducible initialization and validation split.
verbose	Print the loss every 50 epochs.

Value

A list of class "nn_control".

See Also

rem()

---

nn_uncertainty *Bootstrap uncertainty for the neural rem() backend*

Description

Quantifies uncertainty for a rem(method = "nn") fit by a **stratum bootstrap**: the case-control strata are resampled with replacement, the network is refit on each resample (reusing the original nn_control() settings, including the training engine), and the spread across refits yields partial-dependence uncertainty bands and a concordance confidence interval. This is the inferential counterpart that the point-prediction nn backend otherwise lacks (coef() returns NULL).

Usage

```
nn_uncertainty(
  object,
  data,
  B = 200L,
  case = NULL,
  stratum = NULL,
  n_grid = 50L,
  level = 0.95,
  seed = NULL
)
```

Arguments

object	A fitted rem() object with method = "nn".
data	The case-control data frame the model was fit on (same columns).
B	Number of bootstrap resamples.
case, stratum	Event-indicator and stratum columns, resolved exactly as in rem() (defaults: the formula's left-hand side, and cumsum(case == 1)).
n_grid	Grid resolution for the partial-dependence curves.
level	Confidence level for the bands and the concordance interval.
seed	Optional integer seed for the resampling.

Details

Each bootstrap partial-dependence curve is centred (its grid-mean removed) before the pointwise quantiles are taken, so the bands describe uncertainty in the *shape* of each effect, not the conditional-logit's unidentified per-stratum offset.

Value

An object of class "nn_uncertainty": a per-feature list of data.frame(x, lo, med, hi) bands, a concordance quantile interval, and the settings B, level. Has print() and plot() methods.

See Also

rem(), nn_control()

plot.nn_uncertainty *Plot partial-dependence uncertainty bands*

Description

Plot partial-dependence uncertainty bands

Usage

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

Arguments

x	An nn_uncertainty() object.
...	Passed to the underlying plot().

Value

x, invisibly.

radoslaw_email *Manufacturing-company email events (Michalski et al. 2014)*

Description

Time-stamped directed emails among employees of a mid-sized manufacturing company over a nine-month period. Sourced from Network Repository as the ia-radoslaw-email dataset.

Usage

```
radoslaw_email
```

Format

A data frame with 82,927 rows and 4 columns:

time Days since the first email. attr(., "unix_origin") holds the Unix epoch of time = 0.

sender Character employee id.

receiver Character employee id.

weight Integer — 1 for every record in the original file.

Source

Michalski, R., Palus, S., Kazienko, P. (2014). Seed selection for spread of influence in social networks: Temporal vs. static approach. *New Generation Computing* 32(3–4), 213–235. doi:10.1007/s0035401404029. Distributed via https://networkrepository.com/ia-radoslaw-email.php.

`rem` *Fit a relational (hyper)event model on preprocessed case-control data*

Description

`rem()` is the unified front-end for fitting relational event models from **already preprocessed** case-control data (e.g. produced by `eventnet`), where the endogenous/exogenous covariates have already been computed. It is intended to supersede `compare_models()`, `compare_models_smooth()` and `compare_models_global()`, which couple feature computation and fitting.

Usage

```
rem(
  formula,
  data,
  method = c("gam", "clogit", "nn"),
  case = NULL,
  stratum = NULL,
  time = NULL,
  k = NULL,
  gam_method = NULL,
  nn = nn_control(),
  ...
)
```

Arguments

`formula`	A formula; see *Formula syntax*.
`data`	A data.frame of preprocessed case-control data (wide for the `gam` method; long with a case indicator and stratum for `clogit`). For `method = "gam"`, long case-control input (a `event`/`IS_OBSERVED` indicator with control rows) is detected and widened automatically via `widen_case_control()`, with a message.
`method`	Estimation backend; see *Description*.
`case`	Optional name of the 0/1 event-indicator column for the `clogit` and `nn` backends. If `NULL` (default), the indicator is taken from the formula's left-hand side (e.g. `event ~ x`). Ignored by the `gam` method.
`stratum`	Name of the column grouping each case with its controls (required by `clogit`).
`time`	Name of the time column, required for `tv` / `tvnl` terms.
`k`	Optional integer basis dimension passed to `s()` / `te()`.
`gam_method`	Smoothness-selection method for the `gam` backend, passed to `mgcv::gam()`. Defaults to `NULL`, which uses mgcv's own default (`"GCV.Cp"`) and reproduces the Intro-to-REM tutorial parameterization. Set to `"REML"` for the REML fit used in some papers.
`nn`	An `nn_control()` object with the architecture and training hyper-parameters for `method = "nn"`. Ignored by the other backends.
`...`	Reserved for future use.

Details

Two estimation backends are provided:

`"gam"` Degenerate logistic regression on a case-1-control design (Boschi, Lerner & Wit 2025): the response is a constant 1 and the linear predictor is built from event-minus-control differences. Supports smooth time-varying (`tv`), non-linear (`nl`) and time-varying non-linear (`tvnl`) effects via `mgcv::gam()`.

`"clogit"` Conditional logistic regression on a case-k-control design via `survival::clogit()` (linear terms only). The case/control strata are taken from `stratum`, or derived as `cumsum(case == 1)` when `stratum` is `NULL` (assuming each case is immediately followed by its controls, the eventnet blocked layout).

`"nn"` Flexible conditional-logistic models on the same case-k-control design as `clogit`, trained by (mini-batch) gradient descent on the exact risk-set softmax partial likelihood. Two architectures via `nn_control()`: a multilayer perceptron scoring the full covariate vector jointly (interaction-capable), or an `additive_spline` predictor — per-covariate B-spline expansions fitted by stochastic gradient, the STREAM construction of Filippi-Mazzola & Wit (2024, JRSS-C). No coefficient table; `summary()` reports in-sample (and, with a validation split, held-out) concordance and `plot(type = "pdp")` shows per-feature curves. Pure-R implementation, no extra dependencies.

Value

An object of class `"rem"`: a list with the fitted model (`$fit`), the `method`, the original `formula`, the parsed `terms`, and the number of observations `n`. Has `summary()`, `coef()`, `plot()` and `logLik()` methods.

Formula syntax

The right-hand side lists covariates. A bare name is a **linear** effect; wrap a name to request a smooth effect (`gam` method only):

- `tv(x)` — time-varying linear effect: `s(time, by = d_x)`.
- `nl(x)` — non-linear effect: `s(cbind(x_ev, x_nv), by = c(1, -1))`.
- `tvnl(x)` — time-varying non-linear effect (tensor product).
- `re(x)` — random effect of a grouping factor `x` (e.g. the sender), built from the matched `x_ev` / `x_nv` levels as `s(cbind(x_ev, x_nv), by = cbind(1, -1), bs = "re")`, contributing `f(event_level) - f(control_level)` (following the REM tutorial's species-invasiveness term). Falls back to a single column `x` when `x_ev` / `x_nv` are absent. Identified only when the event and control differ on `x`.

For the `gam` method the left-hand side is ignored (the response is the constant case indicator); for `clogit` the left-hand side names the 0/1 event indicator column (e.g. `event ~ x`), unless `case` is given explicitly.

Column resolution

For a covariate `x`, the event/control difference is taken from column `x`, else `d_x`, else `x_ev - x_nv`. Non-linear terms use `transform_x_ev` / `transform_x_nv` when present (the eventnet spline-transformed covariate), otherwise `x_ev` / `x_nv`. `tvnl` uses `transformed_time` when present. Undirected logs (senders only, no receiver/`TARGET` column) are supported.

See Also

`compare_models_smooth()` (superseded), `simulate_relational_events()` (whose `wide = TRUE` output is a valid input here), `simulate_directed_hyperevents_tvnl()`.

Examples

```
set.seed(1)
w <- simulate_relational_events(
  n_events = 300, senders = paste0("a", 1:12), receivers = paste0("a", 1:12),
  n_controls = 1, endogenous_stats = "reciprocity_count",
  endogenous_effects = c(reciprocity_count = 0.6), wide = TRUE)
fit <- rem(~ reciprocity_count, data = w, method = "gam")
coef(fit)
```

---

sample_non_events *Sample non-events for inference*

Description

Given an observed event log, generate nested case-control data by sampling counterfactual sender–receiver pairs according to predefined strategies.

Usage

```
sample_non_events(
  event_log,
  n_controls = 1,
  scope = c("all", "appearance", "citation"),
  mode = c("two", "one"),
  risk = c("standard", "remove"),
  exclude_pairs = NULL,
  allow_loops = FALSE,
  seed = NULL,
  max_attempts = 1000
)
```

Arguments

event_log	Data frame with columns sender, receiver, and time.
n_controls	Number of non-events (controls) to sample per realized event.
scope	Candidate set definition. "all" uses every actor observed in the data; "appearance" restricts to actors that have appeared in prior events; "citation" matches citation networks where senders are restricted to the papers that debut at the current time and receivers must have appeared earlier.
mode	"one" draws both sender and receiver from the same candidate pool (single-mode). "two" samples sender and receiver from separate pools (two-mode).
risk	Strategy governing the risk set. "standard" (default) keeps all unrealized dyads available across strata, whereas "remove" deletes a dyad from the candidate pool after it has occurred (useful for processes such as species invasions where a pair cannot reoccur). Under "remove", dyads firing at the focal event's own timestamp are also kept out of its control pool (concurrent events are not valid non-events at that instant).
exclude_pairs	Optional two-column data.frame/matrix of (sender, receiver) pairs that are structurally ineligible as controls and must never be sampled (e.g. an alien species' native range, or any dyad forbidden in advance). Columns named sender/receiver are used if present, otherwise the first two columns.
allow_loops	Logical; can sampled non-events have identical sender and receiver?
seed	Optional seed for reproducibility.
max_attempts	Maximum resampling attempts per control before giving up (prevents infinite loops when candidate sets are small).

Value

A data.frame containing the original events (event = 1) and the sampled controls (event = 0), grouped by stratum identifiers.

Examples

```
data(classroom_events)
cc <- sample_non_events(classroom_events, n_controls = 1, seed = 1)
head(cc)
```

---

simulate_actor_covariates

Simulate exogenous actor covariates

Description

Create simple exogenous covariate structures for senders and receivers. The function can return static values (one row per actor) or time-stamped processes (one row per actor and time point) that follow independent AR(1) dynamics.

Usage

```
simulate_actor_covariates(
  senders,
  receivers,
  covariate_names,
  time_points = NULL,
  sd = 1,
  rho = 0,
  seed = NULL
)
```

Arguments

`senders`	Character vector of sender actors.
`receivers`	Character vector of receiver actors.
`covariate_names`	Character vector naming the covariates to simulate.
`time_points`	Optional numeric vector of strictly increasing time stamps for time-varying covariates. When omitted, static covariates are returned.
`sd`	Standard deviation of the innovation noise.
`rho`	AR(1) coefficient used when `time_points` is supplied. Must be in (-1, 1).
`seed`	Optional integer to make the simulation reproducible.

Value

A list with two elements: `sender_covariates` and `receiver_covariates`. Each element is either a wide data.frame (static case) or a tidy data.frame with columns `actor`, `time`, `covariate`, and `value` (dynamic case).

Examples

```
sender_cov <- simulate_actor_covariates(
  senders = letters[1:3],
  receivers = LETTERS[1:2],
  covariate_names = c("activity", "recency"),
  time_points = seq(0, 4),
  rho = 0.6,
  sd = 0.2,
  seed = 123
)
str(sender_cov)
```

`simulate_directed_hyperedge_events` *Simulate directed two-mode hyperedge events*

Description

Generates a sequence of *directed* hyperevents from a sender set $I_m \subseteq V^I$ to a receiver set $J_m \subseteq V^J$, with both I_m and J_m non-empty. This is the directed two-mode counterpart to `simulate_hyperedge_events()` and matches the data model used in Boschi, Lerner & Wit (2025) Section 5 for citation networks (authors citing papers).

Usage

```
simulate_directed_hyperedge_events(
  n_events,
  senders,
  receivers,
  min_size_I = 1L,
  max_size_I = 1L,
  min_size_J = 1L,
  max_size_J = 1L,
  baseline_rate = 1,
  endogenous_stats = character(0),
  endogenous_effects = numeric(0),
  start_time = 0
)
```

Arguments

`n_events`	Number of events to simulate.
`senders`	Character vector of sender names V^I.
`receivers`	Character vector of receiver names V^J. Must be non-empty.
`min_size_I`, `max_size_I`	Sender-side cardinality bounds.
`min_size_J`, `max_size_J`	Receiver-side cardinality bounds.
`baseline_rate`	Multiplicative baseline (λ_0).
`endogenous_stats`	Character vector of supported stat names: `"size_I"` (sender-side size penalty), `"size_J"` (receiver-side), `"activity"` (number of past events covering the full focal `(I, J)`), `"subrep_<rho>_<l>"` (directed subset repetition, paper eq. 4).
`endogenous_effects`	Numeric vector of coefficients, same length and order as `endogenous_stats`.
`start_time`	Simulation start time.

Details

At each step the simulator enumerates every candidate hyperedge (I, J) with $|I| \in [\text{min\_size\_I}, \text{max\_size\_I}]$ and $|J| \in [\text{min\_size\_J}, \text{max\_size\_J}]$, computes the rate

$$\lambda(t, I, J) \;=\; \text{baseline\_rate} \;\cdot\; \exp\left(\sum_k \beta_k \, x_k(t, I, J)\right),$$

and draws one event proportional to its rate. The waiting time is exponential with rate equal to the total intensity.

Candidate-space size is exponential in $|V^I|$ and $|V^J|$, so practical use is limited to small actor / item universes.

Value

A directed hyperedge log (`amorem_hyperedge_log` data frame with `I`, `J`, `time` columns; `J` non-empty on every row).

References

Boschi M, Lerner J, Wit EC (2025). *Beyond Linearity and Time-Homogeneity: Relational Hyper Event Models with Time-Varying Non-Linear Effects*. arXiv:2509.05289, Section 5.

Examples

```
hl <- simulate_directed_hyperedge_events(
  n_events  = 40,
  senders   = paste0("a", 1:4),
  receivers = paste0("p", 1:4),
  max_size_I = 2, max_size_J = 2,
  baseline_rate = 0.3,
  endogenous_stats   = c("subrep_1_1", "size_I"),
  endogenous_effects = c(subrep_1_1 = 0.8, size_I = -0.4))
```

`simulate_directed_hyperevents_tvnl` *Simulate directed hyper-events with time-varying and non-linear effects*

Description

A teaching-oriented simulator for **directed relational hyper-events** in which the sender set and the receiver set are disjoint, driven by exogenous group covariates with a **time-varying** effect on the sender side and a **non-linear** effect on the receiver side. It is the packaged, parameterised form of the workshop running example (`sunbelt-workshop-materials/running_example.R`) and produces a ready-to-fit case-control dataset for GAM-based estimation of smooth (TV / NL) effects.

Usage

```
simulate_directed_hyperevents_tvnl(
  sender_attr,
  receiver_attr,
  time_varying_effect = function(t) sin(2 * t),
  nonlinear_effect = function(x) -4 + 2 * exp(-((x - 3)^2)/(2 * 2^2)),
  horizon = 2,
  dt = 0.01,
  n_controls = 1L,
  max_group_size_sender = length(sender_attr),
  max_group_size_receiver = length(receiver_attr)
)
```

Arguments

`sender_attr`	Named numeric vector of sender actor attributes (names are the sender ids).
`receiver_attr`	Named numeric vector of receiver actor attributes.
`time_varying_effect`	Function of one argument giving the time-varying coefficient $\alpha(t)$ multiplying the sender-group covariate. Defaults to `function(t) sin(2 * t)`.
`nonlinear_effect`	Function of one argument giving the non-linear effect $f(x)$ of the receiver-group covariate. Defaults to a Gaussian bump.
`horizon`	Positive numeric; the simulation end time (start is 0).
`dt`	Positive numeric; the time-grid step used by the thinning scheme (must be smaller than `horizon`).
`n_controls`	Non-negative integer; number of non-event pairs sampled per event. Defaults to 1 (a case-1-control design).
`max_group_size_sender`, `max_group_size_receiver`	Integers; the largest subset size considered when enumerating sender / receiver groups. Default to the full actor sets (all non-empty subsets). Use 1 for ordinary dyadic events.

Details

Each sender (resp. receiver) *group* is a non-empty subset of the sender (resp. receiver) actors – a hyperedge endpoint. A group's covariate is the mean of its members' actor attributes. For an ordered group pair (g_s, g_r) the instantaneous rate at time t is

$$\lambda_{g_s,g_r}(t) = \exp\big(\alpha(t)\, x_{g_s} + f(x_{g_r})\big),$$

where $\alpha(t)$ is the time-varying sender effect (`time_varying_effect`), $f(\cdot)$ is the non-linear receiver effect (`nonlinear_effect`), and x denotes a group covariate. Events are drawn on a fixed time grid of width `dt` by a thinning scheme: within each step the next inter-event time is sampled from an exponential with the total rate evaluated at the step midpoint, and the firing pair is chosen with probability proportional to its rate. For every realised event, `n_controls` non-event pairs are sampled uniformly from the remaining group pairs, yielding a case-`n_controls`-control design.

Setting `max_group_size_sender = 1` and `max_group_size_receiver = 1` reduces the groups to single actors, i.e. ordinary directed dyadic events.

Value

A long-format data.frame, one row per (event or control), with columns `event_id` (links a case to its controls), `event_time`, `event` (1 = realised event, 0 = sampled non-event), `sender_group`, `receiver_group` (group labels), and `cov_sender`, `cov_receiver` (group covariates). The true data-generating effect functions are attached as `attr(x, "truth")` (a list with `time_varying_effect`, `nonlinear_effect`, and `horizon`) for comparison against fitted smooths.

Examples

```
set.seed(1234)
sa <- setNames(rnorm(4, 5, 1.5), paste0("S", 1:4))
ra <- setNames(rnorm(4, 3, 2.0), paste0("R", 1:4))
d  <- simulate_directed_hyperevents_tvnl(sa, ra, horizon = 2, n_controls = 1)
head(d)
table(d$event)
```

`simulate_hyperedge_events` *Simulate undirected hyperedge events (multi-actor meetings)*

Description

Generates a sequence of *undirected* hyperevents — meetings of varying size drawn from the actor set `actors` — under a linear hyperedge model. Mirrors the simulation setup of Boschi, Lerner & Wit (2025) Section 4: each event is a subset of `actors` with size in `1..max_size`, fired with rate

$$\lambda(t, I) \;=\; \text{baseline\_rate} \;\cdot\; \exp\!\left(\sum_{k} \beta_k \, x_k(t, I)\right),$$

where each $x_k(t, I)$ is one of the hyperedge-native covariates supported by `hyperedge_features()` (`activity`, `subrep_<rho>` for undirected events) or `size` (the event's cardinality $|I|$).

Usage

```
simulate_hyperedge_events(
  n_events,
  actors,
  max_size,
  baseline_rate,
  endogenous_stats = character(0),
  endogenous_effects = numeric(0),
  start_time = 0,
  min_size = 1L
)
```

Arguments

`n_events`	Number of events to simulate.
`actors`	Character vector of actor names.
`max_size`	Maximum allowed meeting size (w in the paper). Must be in `1..length(actors)`.
`baseline_rate`	Multiplicative baseline (λ_0).
`endogenous_stats`	Character vector of stat names accepted by `hyperedge_features()` (undirected variants — `activity`, `subrep_1`, `subrep_2`, ...) or the literal `"size"` (the event's cardinality).
`endogenous_effects`	Numeric vector of coefficients, same length and order as `endogenous_stats`.
`start_time`	Simulation start time.
`min_size`	Minimum allowed meeting size. Defaults to 1.

Details

At each step the simulator enumerates **every subset** of `actors` with size in `1..max_size`. The per-event work is therefore $O\left(\sum_{s=1}^{w} \binom{|V|}{s}\right)$; practical for small actor counts (e.g. $|V| \leq 20$, `max_size` <= 4).

Value

A hyperedge log (see `hyperedge_log()`) with `n_events` rows.

References

Boschi M, Lerner J, Wit EC (2025). *Beyond Linearity and Time-Homogeneity: Relational Hyper Event Models with Time-Varying Non-Linear Effects*. arXiv:2509.05289.

Examples

```
# Five-actor meetings of size up to 3, with weak attractor on
# repeated triads and a size penalty:
hl <- simulate_hyperedge_events(
  n_events = 50,
  actors   = LETTERS[1:5],
  max_size = 3,
  baseline_rate = 0.2,
  endogenous_stats   = c("subrep_2", "size"),
  endogenous_effects = c(subrep_2 = 0.5, size = -0.3))
```

simulate_relational_events *Simulate relational event sequences*

Description

Generate a simple relational event log for a sender set and receiver set using a softmax allocation rule over dyadic intensities. The process follows the Gillespie algorithm, where the time between events is drawn from an exponential distribution with rate equal to the sum of all dyadic intensities.

Usage

```
simulate_relational_events(
  n_events,
  senders,
  receivers,
  baseline_rate = 1,
  start_time = 0,
  horizon = Inf,
  contribution_logits = NULL,
  sender_covariates = NULL,
  sender_effects = NULL,
  receiver_covariates = NULL,
  receiver_effects = NULL,
  allow_loops = FALSE,
  n_controls = 0,
  endogenous_stats = NULL,
  endogenous_effects = NULL,
  global_covariates = NULL,
  global_effects = NULL,
  method = c("gillespie", "tau_leap"),
  tau = NULL,
  half_life = NULL,
  risk = c("standard", "remove"),
  wide = FALSE
)
```

Arguments

n_events	Number of events to generate.
senders	Character vector listing the sender set $\mathcal{S}$.
receivers	Character vector listing the receiver set $\mathcal{R}$.
baseline_rate	Positive scalar. A constant baseline hazard multiplier applied to all dyads. Defaults to 1.
start_time	Initial time stamp.

`horizon` Optional maximum horizon; simulation stops once the cumulative time would exceed this value.

`contribution_logits`
Optional `length(senders)` x `length(receivers)` matrix of dyad-level contributions to the log-rate (i.e. the dyad-specific part of the linear predictor, distinct from the baseline hazard). Defaults to zeros.

`sender_covariates`
Optional numeric data.frame/matrix with one row per sender.

`sender_effects` Optional numeric vector of coefficients for `sender_covariates`. Required when sender covariates are supplied.

`receiver_covariates`
Optional numeric data.frame/matrix with one row per receiver.

`receiver_effects`
Optional numeric vector of coefficients for `receiver_covariates`. Required when receiver covariates are supplied.

`allow_loops` Logical; whether sender and receiver can coincide.

`n_controls` Integer; number of non-events (controls) to sample uniformly at random for each realized event. If `n_controls > 0`, the function returns a case-control data frame suitable for conditional logistic regression / GAM modeling. Defaults to 0.

`endogenous_stats`
Optional character vector of endogenous mechanisms to include in the rate. Each entry updates a state matrix after every event so the intensity of the next event depends on the realized history. Supported values:

- `"reciprocity_count"` — number of past reverse-dyad events.
- `"reciprocity_binary"` — 1 if the reverse dyad has fired at least once, 0 otherwise.
- `"reciprocity_exp_decay"` — sum of past reverse-dyad events with exponential half-life decay (requires `half_life`).
- `"transitivity_exp_decay"` — $\sum_k e^{-(t-t_{\text{form}}^{(s,k,r)})\log 2/T}$ where $t_{\text{form}}^{(s,k,r)}$ is the formation time of two-path $s \to k \to r$ (definition $t^{(5c)}$ of Juozaitienė & Wit, 2024). Requires `half_life`.
- `"transitivity_exp_decay_ordered"` — same as `"transitivity_exp_decay"` but only counts *ordered* two-paths (s → k strictly before k → r), definition $t^{(6c)}$. Requires `half_life`.
- `"cyclic_exp_decay"` — exp-decayed sum over cyclic two-paths $r \to k \to s$ (paper $c^{(5c)}$). Requires `half_life`.
- `"sending_balance_exp_decay"` — exp-decayed sum over shared-target two-paths $s \to k,\ r \to k$ (paper $sb^{(5c)}$). Requires `half_life`.
- `"receiving_balance_exp_decay"` — exp-decayed sum over shared-source two-paths $k \to s,\ k \to r$ (paper $rb^{(5c)}$). Requires `half_life`.
- `"transitivity_time_recent_interrupted"` / `"transitivity_time_first_interrupted"` — *interrupted* timing variants of the transitivity family (paper $t^{(7ai)}/t^{(7bi)}$). Every $s \to r$ event resets the firing dyad's interrupted state to `NA`, so the value at (s, r) reflects the most recent / first two-path $s \to k \to r$ formed *since the most recent closure event*.

- `"cyclic_time_recent_interrupted"` / `"cyclic_time_first_interrupted"` — same pattern for cyclic two-paths $r \to k \to s$.
- `"sending_balance_time_recent_interrupted"` / `"sending_balance_time_first_interrupte` — same pattern for shared-target two-paths.
- `"receiving_balance_time_recent_interrupted"` / `"receiving_balance_time_first_interr` — same pattern for shared-source two-paths.
- `"reciprocity_time_recent"` — elapsed time since the most recent reverse-dyad event $t - t_{\text{recent}}(r, s)$; reports `0` for dyads whose reverse has never fired (rather than the post-hoc `NA`, so the rate computation stays numeric).
- `"reciprocity_time_first"` — elapsed time since the *first* reverse-dyad event $t - t_{\text{first}}(r, s)$; same `0`-for-never-seen convention.
- `"reciprocity_binary_interrupted"` / `"reciprocity_count_interrupted"` / `"reciprocity_exp_decay_interrupted"` / `"reciprocity_time_recent_interrupted"` / `"reciprocity_time_first_interrupted"` — the *interrupted* reciprocity family of Juozaitienė & Wit (2024) §2.1.3 (definitions $r^{(1i)}, r^{(2i)}, r^{(3i)}, r^{(4ai)}, r^{(4bi)}$). Each variant measures the same quantity as its continuous counterpart but considers only those reverse-dyad events that occurred *since* the most recent same-direction $s \to r$ event. Firing $s \to r$ resets the interrupted state for dyad (s, r).
- `"recency"` — elapsed time on the same ordered dyad $t - t_{\text{last}}(s, r)$, defaulting to $t -$ start_time for dyads that have never fired.
- `"sender_outdegree"` — total number of events previously sent by s (constant across receivers).
- `"receiver_indegree"` — total number of events previously received by r (constant across senders).
- `"transitivity_count"` / `"transitivity_binary"` — number of intermediaries k (or indicator that at least one exists) for which both (s, k) and (k, r) have fired.
- `"cyclic_count"` / `"cyclic_binary"` — number of intermediaries k (or indicator) for which both (r, k) and (k, s) have fired (cyclic two-path closing $s \to r$).
- `"sending_balance_count"` / `"sending_balance_binary"` — number of shared targets k (or indicator) where both (s, k) and (r, k) have fired.
- `"receiving_balance_count"` / `"receiving_balance_binary"` — number of shared sources k (or indicator) where both (k, s) and (k, r) have fired.
- `"transitivity_time_recent"` — elapsed time since the most recent two-path $s \to k \to r$ was completed, for any intermediary k (definition 7ac of Juozaitienė & Wit, 2024). Reports `0` for dyads where no two-path has ever existed.
- `"transitivity_time_first"` — elapsed time since the *first* two-path $s \to k \to r$ was completed (definition 7bc of Juozaitienė & Wit, 2024). Same `0`-for-never-seen convention.
- `"cyclic_time_recent"` / `"cyclic_time_first"` — elapsed time since the most recent / first cyclic two-path $r \to k \to s$ was completed.

- `"sending_balance_time_recent"` / `"sending_balance_time_first"` — elapsed time since the most recent / first shared-target two-path $s \to k,\ r \to k$ was completed.
- `"receiving_balance_time_recent"` / `"receiving_balance_time_first"` — elapsed time since the most recent / first shared-source two-path $k \to s,\ k \to r$ was completed.
- `"transitivity_count_ordered"` / `"transitivity_binary_ordered"` — number of intermediaries k (or indicator) for which an ordered two-path $s \to k$ *before* $k \to r$ has been observed (definitions $t^{(4c)}$ / $t^{(2c)}$ of Juozaitienė & Wit, 2024).
- `"transitivity_time_recent_ordered"` / `"transitivity_time_first_ordered"` — elapsed time since the most recent / first ordered two-path $s \to k$ *before* $k \to r$ was completed (definitions $t^{(8ac)}$ / $t^{(8bc)}$ of Juozaitienė & Wit, 2024).

Defaults to `NULL` for a memoryless process.

`endogenous_effects`
: Numeric vector of linear coefficients for `endogenous_stats`. May be named (names must match `endogenous_stats`) or unnamed (positionally matched). Required when `endogenous_stats` is supplied.

`global_covariates`
: Optional data.frame describing piecewise-constant global covariates: variables whose value at time t is the same for every dyad (e.g. weekday/weekend, weather, policy regime). Must contain a numeric `time_start` column giving the start of each interval; rows are assumed sorted in time and the first `time_start` must be at or before `start_time`. Each additional numeric column is treated as a global covariate. Defaults to `NULL` (no global effects).

`global_effects`
: Numeric vector of linear coefficients for the global covariates. May be named (names must match the covariate columns in `global_covariates`) or unnamed (positionally matched). Required when `global_covariates` is supplied.

`method`
: Simulation algorithm. Either `"gillespie"` (the default, exact event-driven algorithm: draw inter-event waiting times one at a time) or `"tau_leap"` (approximate, time-driven algorithm: advance the clock in fixed `tau` increments and Poisson-sample event counts per dyad within each step).

`tau`
: Positive scalar; the step size for `method = "tau_leap"`. Required when method is `"tau_leap"` and ignored otherwise. Smaller values give better approximation but more iterations; as $\tau \to 0$ the tau-leap result converges in distribution to the exact Gillespie result.

`half_life`
: Positive scalar; the half-life T (in time units) used by every `*_exp_decay` stat. A past contribution at time t_k carries weight $\exp(-(t - t_k)\ \log 2/T)$ into the stat value at time t. The same T is shared across all decay stats, matching the convention in Juozaitienė & Wit (2024). Required when any of `"reciprocity_exp_decay"`, `"transitivity_exp_decay"`, `"transitivity_exp_decay_ordered"` is in `endogenous_stats`.

`risk`
: Risk-set rule. `"standard"` (the default) keeps every dyad eligible at every step. `"remove"` removes a dyad from the risk set as soon as it fires, which mimics one-shot processes such as species invasions or first-citation events.

`wide` Logical; when `TRUE` (which requires `n_controls = 1`), the result is returned in a wide case-1-control format with one row per event instead of the default long format. Each row carries the event-dyad actors (`sender_ev`, `receiver_ev`), the matched non-event-dyad actors (`sender_nv`, `receiver_nv`), and, for every covariate column, the event value (`<cov>_ev`), the control value (`<cov>_nv`), and their difference (`d_<cov>`, event minus control). Defaults to `FALSE`.

Details

When `global_covariates` is supplied, the simulator uses a boundary-aware Gillespie scheme: the total event rate is rescaled by $\exp(\sum_k \beta_k \, x_k(t))$; whenever a sampled waiting time would cross an interval boundary, the clock is advanced to the boundary without recording an event, and the next waiting time is redrawn under the new global multiplier. Global covariates do not change the per-dyad selection probabilities (the multiplier cancels), only the waiting-time distribution. When combined with `endogenous_stats`, the per-dyad rates are recomputed at every step from the current endogenous state and then rescaled by the global multiplier.

The `"tau_leap"` algorithm advances the clock by a user-chosen step τ and draws, for every dyad, a $\mathrm{Poisson}(\lambda_{sr}(t)\,\tau)$ number of events using the rates at the *start* of the step. Multiple events can fire in the same step; they are placed at uniform times within $[t, t+\tau)$ and reported in time order, but they share the start-of-step endogenous state and global multiplier. Endogenous state is updated once at the end of the step using all events in that step. The tau-leap algorithm trades exactness for predictable, vectorised work per step; it is most useful for high-rate regimes or for problems where the per-event recomputation in the Gillespie path is the bottleneck. Choose τ small enough that (i) $\lambda\tau \ll 1$ on every active dyad and (ii) τ is smaller than the shortest interval in `global_covariates` (within-step boundary crossings are not resolved; the start-of-step global multiplier is used for the entire step).

Value

If `n_controls = 0`, a data.frame with columns `sender`, `receiver` and `time`. If `n_controls > 0`, it returns a long-format data.frame with additional columns `stratum` (grouping an event with its controls) and `event` (1 for the realized event, 0 for controls). When `endogenous_stats` is supplied, one extra column per stat is appended carrying the value each row's dyad had at its event time (immediately before the event fired), so downstream conditional logistic / GAM estimators can recover the effects. When `global_covariates` is supplied, one column per covariate is appended carrying the value of that covariate at each row's event time. When `wide = TRUE` (which requires `n_controls = 1`), this long case-control output is reshaped to one row per event with columns `stratum`, `time`, `sender_ev`, `receiver_ev`, `sender_nv`, `receiver_nv` and, for each covariate, `<cov>_ev`, `<cov>_nv` and `d_<cov>`.

Examples

```
set.seed(1)
senders <- receivers <- LETTERS[1:3]
sender_cov <- data.frame(activity = c(0.5, -0.2, 1.1))
receiver_cov <- data.frame(popularity = c(0.1, 0.3, -0.4))
# Standard event simulation
events <- simulate_relational_events(
  n_events = 5,
```

```
  senders = senders,
  receivers = receivers,
  sender_covariates = sender_cov,
  sender_effects = 1,
  receiver_covariates = receiver_cov,
  receiver_effects = 2
)
events

# Case-control generation for partial likelihood inference
cc_events <- simulate_relational_events(
  n_events = 5,
  senders = senders,
  receivers = receivers,
  sender_covariates = sender_cov,
  sender_effects = 1,
  n_controls = 2
)
head(cc_events)

# Ready-made case-1-control (wide) dataset for degenerate logistic regression
wide_events <- simulate_relational_events(
  n_events = 5,
  senders = senders,
  receivers = receivers,
  n_controls = 1,
  endogenous_stats = "reciprocity_count",
  endogenous_effects = c(reciprocity_count = 0.6),
  wide = TRUE
)
head(wide_events)
```

`social_evolution_actors` *Actor attributes for the Social Evolution study*

Description

Per-actor covariates for social_evolution_calls and social_evolution_friendship.

Usage

```
social_evolution_actors
```

Format

A data frame with 84 rows and 4 columns:

id Character actor id (`"Actor 1"`, `"Actor 2"`, ...).

present Logical — whether the actor was present at the start of the study window.

floor Integer dormitory floor.

gradeType Factor — student grade type (freshman, sophomore, junior, senior, graduate-tutor).

Source

Madan et al. (2011), via `goldfish`. See social_evolution_calls.

`social_evolution_calls`

Phone calls in the Social Evolution study (Madan et al. 2011)

Description

Time-stamped directed phone calls among undergraduates in an MIT residence hall over the 2008–2009 academic year. Sourced from the `goldfish` R package (`Social_Evolution$calls`).

Usage

```
social_evolution_calls
```

Format

A data frame with 439 rows and 4 columns:

time Days since the first recorded call. `attr(., "unix_origin")` holds the Unix epoch of `time = 0`.

sender Character actor id matching social_evolution_actors`$id`.

receiver Same domain as `sender`.

increment Integer increment recorded for the call (typically 1).

Source

Madan, A., Cebrian, M., Moturu, S., Farrahi, K. (2011). Sensing the "health state" of a community. *IEEE Pervasive Computing* 11(1), 36–45. doi:10.1109/MPRV.2011.79. Redistributed via the `goldfish` R package (github.com/snlab-ch/goldfish), dataset `Social_Evolution`.

See Also

social_evolution_actors, social_evolution_friendship

social_evolution_friendship *Friendship-survey events for the Social Evolution study*

Description

Self-reported friendship ties recorded at survey waves throughout the Social Evolution study.

Usage

```
social_evolution_friendship
```

Format

A data frame with 766 rows and 4 columns:

time Days since the first recorded call (same origin as social_evolution_calls). `attr(., "unix_origin")` holds the Unix epoch of `time = 0`.

sender Character actor id (the survey respondent).

receiver Character actor id (the nominated friend).

replace Integer — `1` adds the tie, `0` removes it.

Source

Madan et al. (2011), via `goldfish`.

See Also

social_evolution_calls, social_evolution_actors

standardize_event_log *Standardize a relational event log*

Description

Module A focuses on preprocessing utilities. This helper normalizes user supplied event logs into the canonical `sender`/`receiver`/`time` structure expected elsewhere in the package. It also handles common cleaning tasks such as sorting, dropping missing rows, and removing loops.

Usage

```
standardize_event_log(
  event_log,
  sender_col = "sender",
  receiver_col = "receiver",
  time_col = "time",
  sort = TRUE,
  drop_nas = TRUE,
  drop_loops = FALSE,
  strictly_increasing_time = FALSE,
  remove_duplicates = TRUE,
  keep_extra = TRUE
)
```

Arguments

`event_log`	A data.frame (or tibble) containing at least one row per event.
`sender_col`, `receiver_col`, `time_col`	Column names storing the sender, receiver, and time information.
`sort`	Logical; should the output be sorted by time (ties are kept in input order)?
`drop_nas`	Logical; if `TRUE`, rows with missing sender/receiver/time are removed. Otherwise an error is thrown when NAs are present.
`drop_loops`	Logical; when `TRUE`, self-loops (`sender == receiver`) are dropped.
`strictly_increasing_time`	Logical; if `TRUE`, an error is raised when non-increasing time stamps are detected after sorting.
`remove_duplicates`	Logical; drop duplicated combinations of sender/receiver/time.
`keep_extra`	Logical; if `FALSE`, only the standardized columns are returned. When `TRUE`, additional columns from the original input are preserved.

Value

A data.frame with columns `sender`, `receiver`, and `time`. The return object is tagged with class `"amorem_event_log"` for downstream dispatch.

Examples

```
data(classroom_events)
std <- standardize_event_log(classroom_events)
head(std)
```

`transform_recency` *Recency transform of inter-event time gaps*

Description

Maps non-negative time gaps δ to bounded recency weights via

$$w(\delta) \;=\; \exp\!\Big(-\frac{\delta}{2\,m}\Big),$$

where m is the median of the supplied (or reference) gaps. Large gaps map toward 0; gaps near 0 map toward 1. The half-life of the kernel is $2m \log 2$, so the median gap itself is mapped to approximately $e^{-1/2} \approx 0.607$.

Usage

```
transform_recency(delta, half_life = NULL, reference = NULL)
```

Arguments

`delta`	Numeric vector of non-negative time gaps. NAs propagate.
`half_life`	Optional positive scalar. If supplied, used directly as the kernel scale $2m$, bypassing the median rule.
`reference`	Optional numeric vector. If supplied, the median is computed on `reference` instead of `delta`. Useful when transforming new data using a scale fitted on training data.

Details

This is the data-driven recency parametrisation used as a preprocessing step for global and exogenous covariates in Lembo, Juozaitiene, Vinciotti & Wit (2025) and matches the "recency" axis of `endogenous_features()`.

Value

Numeric vector the same length as `delta`, with values in `(0, 1]`. NAs in `delta` are preserved.

References

Lembo M, Juozaitiene R, Vinciotti V, Wit EC (2025). *Relational Event Models with Global Covariates*. JRSS-C.

Examples

```
set.seed(1)
gaps <- rexp(20, rate = 0.5)
transform_recency(gaps)
transform_recency(gaps, half_life = 1)
```

`widen_case_control` *Convert a long case-control event log to wide case-1-control format*

Description

Reshapes a long case-(k-)control dataset – one row per case and per control, with a 0/1 case indicator – into a wide **case-1-control** table with one row per case. For each covariate the event value (<cov>_ev), the matched control value (<cov>_nv) and their difference (d_<cov>, event minus control) are emitted, ready for the gam backend of `rem()`.

Usage

```
widen_case_control(
  data,
  case = NULL,
  stratum = NULL,
  covariates = NULL,
  control_index = 1L,
  keep_ids = TRUE
)
```

Arguments

`data`	A long case-control data.frame.
`case`	Optional name of the 0/1 event-indicator column. If `NULL` (default), it is auto-detected from the package's `event` column (as produced by `sample_non_events()`) or eventnet's `IS_OBSERVED`, preferring `event` when both are present.
`stratum`	Optional name of the column grouping each case with its controls. When `NULL`, the stratum is derived as `cumsum(case == 1)` (assuming each case is immediately followed by its controls).
`covariates`	Character vector of covariate columns to widen. When `NULL`, all numeric columns are used except the case indicator, the stratum, and the standard eventnet bookkeeping columns (`EVENT`, `INTEGER_TIME`, `TIME_POINT`, `TIME_UNIT`, `EVENT_INTERVAL`).
`control_index`	Which control within each stratum to pair with the case (default the first). Lets a case-k-control log be reduced to case-1-control.
`keep_ids`	Logical; when `TRUE` (default) the sender/receiver identifier columns present in `data` are carried into the output as `sender_ev` / `receiver_ev` (the observed event) and `sender_nv` / `receiver_nv` (the matched control), so the dyads behind each case-control pair remain recoverable (and become available to `re()` grouping terms in `rem()`). Set to `FALSE` to emit only the widened covariate columns.

Details

This is the preprocessing companion to `rem()` for eventnet-style output, where a case row is followed by its controls and the stratum id is left blank on control rows.

Value

A data.frame with one row per case: a `stratum` column, the sender/receiver identifiers (`sender_ev`/`receiver_ev`/`sender_nv`/`receiver_nv`, when present in `data` and `keep_ids = TRUE`) and, for each covariate, `<cov>_ev`, `<cov>_nv` and `d_<cov>`. Strata without exactly one case or without the requested control are dropped (with a message).

See Also

`rem()`, `simulate_relational_events()` (`wide = TRUE`).

Examples

```
set.seed(1)
long <- data.frame(
  IS_OBSERVED = rep(c(1, 0, 0), 4),
  x = rnorm(12), y = rnorm(12))
widen_case_control(long, control_index = 1)
```

Index