Package 'redeem'

June 22, 2026

Type Package

Title Relational Event and Durational Event Models

Version 1.0.0

Date 2026-06-18

Author Cornelius Fritz [aut, cre]

Maintainer Cornelius Fritz <corneliusfritz2010@gmail.com>

Description Model relational and durational events in a
counting process framework, with functions for estimating and simulating
Relational Event Models (REM) and Durational Event Models (DEM). Includes
support for time-varying covariates, windowed statistics, and
high-dimensional node-level fixed effects. References include
Fritz et al. (2026) ``Scalable Durational Event Models: Application to
Physical and Digital Interactions'' <doi:10.48550/arXiv.2504.00049>.

License GPL-3

Depends R (>= 4.0.0)

Imports Rcpp (>= 1.0.12), data.table, MASS, survival, stats,
grDevices, graphics, utils, digest

LinkingTo Rcpp, RcppProgress, RcppArmadillo

Suggests testthat (>= 3.0.0), network, knitr, rmarkdown

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

URL https://corneliusfritz.github.io/redeem/

BugReports https://github.com/corneliusfritz/redeem/issues

Config/roxygen2/version 8.0.0

NeedsCompilation yes

Repository CRAN

Date/Publication 2026-06-22 16:20:07 UTC

Contents

`check_matrix` *Validate the Structure of a Durational Event List*

Description

This function checks the validity of a dyadic interaction matrix by ensuring that each interaction start event has a corresponding end event, that no interactions overlap within a dyad, and that no missing values are present.

Usage

```
check_matrix(df, return_matrix = FALSE, start_time = NULL)
```

Arguments

`df`	A data frame with at least four columns, representing events, where: **Column 1** Event timing or ID (e.g., timestamp). **Column 2** "From" node ID for the dyadic interaction. **Column 3** "To" node ID for the dyadic interaction. **Column 4** Event type (1 for start, 0 for end of interaction).
`return_matrix`	Logical; if TRUE, returns the (potentially repaired) event matrix. Defaults to FALSE, in which case the function returns `TRUE` if the matrix is valid.
`start_time`	Numeric; optional reference time for adding missing start events. Defaults to NULL, in which case the earliest time in the data is used.

Details

The function performs the following checks:

- Missing values: If any are found, a warning is issued and FALSE is returned.
- Interaction pairing: Each start event (1) must have a corresponding end event (0) without overlap.
- Non-overlapping intervals: Ensures that no start event occurs while another interaction is active.

Value

Logical; TRUE if all interactions are valid, FALSE otherwise. If the data contains missing values, the function issues a warning and returns FALSE.

Examples

```
# Create a valid event matrix with durational events (start=1, end=0)
df <- matrix(c(
  1.0, 1, 2, 1,
  2.0, 1, 2, 0,
  1.5, 3, 4, 1,
  3.0, 3, 4, 0
), ncol = 4, byrow = TRUE)
colnames(df) <- c("time", "from", "to", "type")

# Check if the event matrix is valid
check_matrix(df)
```

---

control.redeem *Control Parameters for REDEEM Models*

Description

Unified control object to manage estimation parameters for rem and dem functions.

Usage

```
control.redeem(
  it_max = 100,
  tol = 1e-10,
  accelerated = FALSE,
  verbose = FALSE,
  weighting = TRUE,
  subsample = 1,
  build_time = NULL,
  use_glm = FALSE,
  return_data = FALSE,
```

```
  save_hist = TRUE,
  estimate = "Blockwise",
  legacy = FALSE,
  check_matrix = FALSE,
  inf_unidentifiable = TRUE
)
```

Arguments

it_max	Integer; maximum number of iterations for the algorithm. Defaults to 100.
tol	Numeric; convergence tolerance. Defaults to 1e-10.
accelerated	Logical; if TRUE, uses SQUAREM acceleration for MM updates. Defaults to FALSE.
verbose	Logical; if TRUE, prints progress information. Defaults to FALSE.
weighting	Logical; whether to use weighting to group identical observations. Defaults to TRUE.
subsample	Numeric; proportion of data to subsample for internal GLM checks. Defaults to 1.
build_time	Numeric; time at which to start building the estimation dataset. Events before this time are used to compute statistics but not included as observations. Defaults to NULL, in which case all events are included.
use_glm	Logical; if TRUE, uses standard GLM for updating core coefficients. This is often slower but can yield more robust updates. Defaults to FALSE.
return_data	Logical; whether to return preprocessed data frames in the result. Defaults to FALSE.
save_hist	Logical; whether to save the iteration history of coefficients. Defaults to TRUE.
estimate	Character; estimation method for dem and rem ("Blockwise", "NR", or "GD"). Defaults to "Blockwise".
legacy	Logical; if TRUE, uses a single glm.fit call instead of the iterative loop. Defaults to FALSE.
check_matrix	Logical; whether to apply check_matrix to the event data before estimation. If TRUE, repairs missing events (e.g., adding start events for interactions that only have end events). Defaults to FALSE.
inf_unidentifiable	Logical; whether to set unidentifiable coefficients (e.g., actors with 0 event counts, globally invariant/collinear covariates) to -Inf. Defaults to TRUE.

Value

A list of class "redeem_control" containing the specified parameters.

dem *Durational Event Model (DEM) Estimation*

Description

Estimates a Durational Event Model (DEM) for relational event sequences where interactions have a duration.

Usage

```
dem(
  events,
  training_start = 0,
  exogenous_end = NULL,
  formula_0_1 = NULL,
  formula_1_0 = NULL,
  n_nodes = NULL,
  directed = FALSE,
  estimate_0_1 = NULL,
  estimate_1_0 = NULL,
  coef_0_1 = NULL,
  coef_1_0 = NULL,
  semiparametric = FALSE,
  simultaneous_interactions = TRUE,
  control = control.redeem()
)
```

Arguments

`events`	A matrix of events with columns `time`, `from`, `to`, and `type` (1 for start, 0 for end, 3 for exogenous changes).
`training_start`	Numeric; the time point at which to start the estimation. Defaults to 0.
`exogenous_end`	Numeric; the exogenous time point at which the observational period ends. Defaults to NULL, which implies that time when the final event was observed is taken as the end of the observational period.
`formula_0_1`	A one-sided `formula` specifying the sufficient statistics for the formation process ($0 \rightarrow 1$). The right-hand side must be composed of terms from `redeem_terms`. For example: `~ inertia() + degree`. An intercept (`~ 1`) is the minimal specification. Defaults to NULL.
`formula_1_0`	A one-sided `formula` specifying the sufficient statistics for the dissolution process ($1 \rightarrow 0$). The right-hand side must be composed of terms from `redeem_terms`. An intercept (`~ 1`) is the minimal specification. Defaults to NULL.
`n_nodes`	Integer; the total number of actors in the network. If `NULL` (default), it is automatically identified based on the actors in the `events` set.
`directed`	Logical; whether the interaction events are directed. Defaults to FALSE.

`estimate_0_1` Logical; whether to estimate the formation process. Defaults to NULL, in which case it is estimated if `formula_0_1` is provided.

`estimate_1_0` Logical; whether to estimate the dissolution process. Defaults to NULL, in which case it is estimated if `formula_1_0` is provided.

`coef_0_1` Numeric vector; initial coefficients for the formation model. If provided, this must be a concatenated vector of:

1. Core coefficients: values for sufficient statistics in the formula.
2. Degree coefficients (if `degree` is in the formula): a vector of length `n_nodes` (undirected) or `2 * n_nodes` (directed, sender effects first then receiver effects).
3. Baseline coefficients (if temporal changepoints are present): a vector of length equal to the number of baseline intervals (equal to number of changepoints if an intercept/degree is present, or changepoints + 1 if neither is present).

Defaults to NULL, in which case default starting values are automatically computed.

`coef_1_0` Numeric vector; initial coefficients for the dissolution model. If provided, this must be a concatenated vector of:

1. Core coefficients: values for sufficient statistics in the formula.
2. Degree coefficients (if `degree` is in the formula): a vector of length `n_nodes` (undirected) or `2 * n_nodes` (directed, sender effects first then receiver effects).
3. Baseline coefficients (if temporal changepoints are present): a vector of length equal to the number of baseline intervals (equal to number of changepoints if an intercept/degree is present, or changepoints + 1 if neither is present).

Defaults to NULL, in which case default starting values are automatically computed.

`semiparametric` Logical; whether to use a semiparametric baseline. Defaults to FALSE. See the 'Semiparametric Baseline' section for details.

`simultaneous_interactions` Logical; whether to allow simultaneous interactions (i.e. multiple active events for the same actor or dyad at the same time). Defaults to TRUE.

`control` A list of control parameters from `control.redeem`. Defaults to `control.redeem()`.

Details

The Durational Event Model (DEM) is a general framework for analyzing durational events, extending standard Relational Event Models (REM) by decoupling the modeling of event incidence from event duration. It characterizes the dynamics via two separate continuous-time counting processes:

Formation Process $(0 \to 1)$ Counts the number of times that actor pair (i, j) starts an interaction up to time t. The incidence intensity is denoted by $\lambda_{i,j}^{0\to 1}(t|\mathscr{H}_t)$.

Dissolution Process $(1 \to 0)$ Counts the number of times that actor pair (i, j) stops interacting up to time t. The dissolution intensity is denoted by $\lambda_{i,j}^{1\to 0}(t|\mathscr{H}_t)$.

Under the assumption that the processes are non-homogeneous Poisson processes, the intensities are modeled as:

$$\lambda_{i,j}^{0\to1}(t|\mathscr{H}_t,\theta^{0\to1}) = \exp(s_{i,j}^{0\to1}(\mathscr{H}_t)^\top\alpha^{0\to1} + \beta_i^{0\to1} + \beta_j^{0\to1} + f(t,\gamma^{0\to1}))$$

$$\lambda_{i,j}^{1\to0}(t|\mathscr{H}_t,\theta^{1\to0}) = \exp(s_{i,j}^{1\to0}(\mathscr{H}_t)^\top\alpha^{1\to0} + \beta_i^{1\to0} + \beta_j^{1\to0} + f(t,\gamma^{1\to0}))$$

where:

- $s_{i,j}(\mathscr{H}_t)$ is a vector of dynamic network statistics capturing the history of past interactions $\mathscr{H}_t$.
- α is a parameter vector determining the covariate effects.
- β_i and β_j are actor-specific sociality/popularity parameters (degree correction) capturing actor heterogeneity.
- $f(t,\gamma)$ is a piecewise-constant step function modeling temporal baseline fluctuations across a set of changepoints.

To satisfy the Feller criterion and ensure that the continuous-time counting process remains non-explosive, count-based network statistics (such as inertia or common partners) are typically log-transformed on the $\log(x+1)$ scale.

Value

An object of class `dem_object` containing model estimates, log-likelihoods, and preprocessed data. See `dem_object` for details on the components of the returned object and S3 methods.

Scalable Estimation Algorithm

The likelihood of the model is separable with respect to $\theta^{0\to1}$ and $\theta^{1\to0}$, allowing independent estimation of the incidence and duration components. Traditional maximum likelihood estimation via standard Newton-Raphson requires computing and inverting an $O(N^2)$ Hessian matrix, which is computationally prohibitive for larger networks. To bypass this, the `redeem` package implements a highly scalable block-coordinate ascent algorithm that separates parameter updates:

1. **Step 1**: Update covariate parameters α using a standard Newton-Raphson update.
2. **Step 2**: Update high-dimensional actor popularity baselines β using Minorization-Maximization (MM) steps, avoiding explicit matrix inversion.
3. **Step 3**: Update baseline step function parameters γ via a closed-form step.

More information is provided in Fritz et at. (2026).

Semiparametric Baseline

When `semiparametric = TRUE`, the baseline rates for both the formation ($0 \to 1$) and dissolution ($1 \to 0$) processes are left completely unspecified. Both processes are estimated as separate Cox proportional hazards models using the `survival` package. In this path:

- For the formation process, each start event (1) is treated as a failure, and all inactive dyads at that time constitute the risk set.

- For the dissolution process, each end event (0) is treated as a failure, and all currently active interactions constitute the risk set.
- The exact waiting times (durations of the active and inactive states) are conditioned out, and estimation is based solely on the ordering of events and the relative dyadic intensities at each transition time.
- This approach is highly robust to arbitrary temporal fluctuations in baseline rates since no piecewise-constant temporal baselines or changepoints need to be specified.
- **Limitations**: This path does **not** support the specialized scalable estimation of sender/receiver popularity effects (`degree`) or piecewise-constant temporal baselines.

References

Fritz, C., Rastelli, R., Fop, M., & Caimo, A. (2026). Scalable Durational Event Models: Application to Physical and Digital Interactions. arXiv:2504.00049.

Examples

```
# Simulate some durational data
n <- 20
events <- matrix(c(
  1.2, 1, 5, 1,
  2.5, 1, 5, 0,
  3.1, 2, 8, 1,
  4.4, 2, 8, 0
), ncol = 4, byrow = TRUE)
colnames(events) <- c("time", "from", "to", "type")

# Estimate a simple DEM
fit <- dem(
  events = events,
  n_nodes = n,
  formula_0_1 = ~1,
  formula_1_0 = ~1,
  control = control.redeem(estimate = "Blockwise")
)
summary(fit)
```

`dem.simulate` *Simulate events based on specified formulas and coefficients*

Description

Simulate events based on specified formulas and coefficients

Usage

```
dem.simulate(
  events = matrix(0, 0, 4),
  formula_0_1 = NULL,
  formula_1_0 = NULL,
  coef_0_1 = numeric(0),
  coef_1_0 = numeric(0),
  coef_degree_0_1 = 0,
  coef_degree_1_0 = 0,
  n_events = 0,
  time = 0,
  max_events = 4e+05,
  n_nodes,
  verbose = FALSE,
  baseline_0_1 = NULL,
  baseline_1_0 = NULL,
  simultaneous_interactions = TRUE,
  seed = 123,
  directed = FALSE
)
```

Arguments

`events`	A matrix representing the initial events with columns `time`, `from`, `to`, and `type` (1 for start, 0 for end, 3 for exogenous changes). Defaults to an empty 4-column matrix.
`formula_0_1`	A one-sided `formula` specifying the sufficient statistics for the formation process ($0 \to 1$). The right-hand side must be composed of terms from `redeem_terms`. For example: `~ inertia() + degree`. An intercept (`~ 1`) is the minimal specification. Defaults to NULL.
`formula_1_0`	A one-sided `formula` specifying the sufficient statistics for the dissolution process ($1 \to 0$). The right-hand side must be composed of terms from `redeem_terms`. An intercept (`~ 1`) is the minimal specification. Defaults to NULL.
`coef_0_1`	Numeric vector; coefficients for the formation process ($0 \to 1$). Defaults to an empty numeric vector.
`coef_1_0`	Numeric vector; coefficients for the dissolution process ($1 \to 0$). Defaults to an empty numeric vector.
`coef_degree_0_1`	Numeric; degree coefficient for the formation process ($0 \to 1$). Defaults to 0.
`coef_degree_1_0`	Numeric; degree coefficient for the dissolution process ($1 \to 0$). Defaults to 0.
`n_events`	Integer; number of events to simulate. Defaults to 0.
`time`	Numeric; simulation time limit. Defaults to 0.
`max_events`	Integer; maximum number of total events. Defaults to 400000.
`n_nodes`	Integer; the total number of actors in the network.

verbose	Logical; if TRUE, prints progress information. Defaults to FALSE.
baseline_0_1	Numeric vector; baseline for the 0 to 1 transition. If the formula for this process contains an Intercept or a degree term, then baseline_0_1 should be a numeric vector with length equal to the number of changepoints, representing the shifts in the baseline for each interval after the first. If the formula contains neither, then baseline_0_1 must have length equal to the number of changepoints + 1. Defaults to NULL.
baseline_1_0	Numeric vector; baseline for the 1 to 0 transition. Similar to baseline_0_1, its length depends on whether the 1 to 0 formula contains an intercept or degree term. Defaults to NULL.
simultaneous_interactions	Logical; whether to allow simultaneous interactions (i.e. multiple active events for the same actor or dyad at the same time). Defaults to TRUE.
seed	Integer; random seed for simulation. Defaults to 123.
directed	Logical; whether the interaction events are directed. Defaults to FALSE.

Value

A matrix of simulated events.

Note

Multi-stream event models are currently not supported in simulation.

Examples

```
# Simulate events from a DEM model structure
n <- 10
f_0_1 <- ~ 1 + inertia(transformation = "identity")
f_1_0 <- ~ 1

# Simulating events
evs <- dem.simulate(
  formula_0_1 = f_0_1,
  formula_1_0 = f_1_0,
  n_nodes = n,
  time = 2.0,
  coef_0_1 = c(1.0, 0.5),
  coef_1_0 = c(-0.5),
  seed = 42,
  max_events = 100
)
head(evs)
```

dem_object *The dem Object*

Description

An object of class dem returned by the dem function, representing a fitted Durational Event Model.

Value

A dem object is a list containing the following components:

- call: The matched call.
- event_numbers: A vector containing the number of events.
- model_0_1: The fitted model for transition 0 -> 1 (formation).
- model_1_0: The fitted model for transition 1 -> 0 (dissolution).
- events: The preprocessed event matrix.
- formula_0_1: The formula for transition 0 -> 1.
- formula_1_0: The formula for transition 1 -> 0.
- n_nodes: The number of nodes.
- simultaneous_interactions: Logical indicating whether simultaneous interactions were allowed.
- directed: Logical indicating whether the events are directed.
- training_start: The start time of the training period.
- build_time: The time at which the estimation dataset started building.
- max_time: The maximum event time.
- exogenous_end: The end time of the exogenous period.
- time_changepoints: Time points where baseline intensity changes.
- labels_changepoints: Labels for the time intervals.
- subsample: Subsample proportion used.
- return_data: Logical indicating whether preprocessed data frames were returned.
- runtime: The estimation runtime.
- window_map: The window map used for calculation.
- preprocessed: Preprocessed data structures.

Methods (S3)

The following S3 methods are implemented for dem objects:

- print(x, ...): Prints a brief summary of the DEM model.
  - x: A dem object.
  - ...: Additional arguments passed to printing function.

- `summary(object, ...)`: Summarizes model results, including parameter estimates, standard errors, and fit statistics (AIC/BIC). Returns an object of class `summary.dem`.
  - `object`: A `dem` object.
  - `...`: Additional arguments passed to summary method.
- `plot(x, which = 3, separate = FALSE, baseline = FALSE, ...)`: Generates trace plots for coefficients and log-likelihood histories.
  - `x`: A `dem` object.
  - `which`: Integer indicating transition plots to display. Options are `1` for formation (0 -> 1), `2` for dissolution (1 -> 0), or `3` (default) for both.
  - `separate`: Logical. If `TRUE`, each plot is generated in a new window or as a separate sequence. Defaults to `FALSE`.
  - `baseline`: Logical. If `TRUE`, plots the estimated baseline step function. Defaults to `FALSE`.
  - `...`: Additional arguments passed to the underlying trace plotting method. Supported parameters include:
    - `coefs`: Logical. If `TRUE` (default), plots iteration traces for core/fixed coefficients.
    - `degree`: Logical. If `TRUE` (default), plots iteration traces for degree/actor effects.
    - `time`: Logical. If `TRUE` (default), plots iteration traces for temporal baseline effects.
    - `llh`: Logical. If `TRUE` (default), plots trace of log-likelihood history.
    - `sub_label`: Character. Optional subtitle to display at the bottom of the figure.
- `plot_baseline(x, process = c("formation", "dissolution"), ...)`: Plots the step function of the estimated baseline intensity.
  - `x`: A `dem` object.
  - `process`: Character. Specifies whether to plot the baseline for `"formation"` (0 -> 1) (default) or the `"dissolution"` (1 -> 0) process.
  - `...`: Additional graphical parameters passed to `plot`.
- `predict(object, time = NULL, type = c("response", "lp", "terms"), process = c("both", "formation", "dissolution"), ...)`: Predicts the intensity, linear predictor, or term contributions for a fitted DEM model.
  - `object`: A `dem` object.
  - `time`: Numeric vector; optional time point(s) at which to predict. Defaults to NULL.
  - `type`: Character; the type of prediction. Defaults to `"response"`.
  - `process`: Character; the transition process to predict. Defaults to `"both"`.
  - `...`: Additional arguments.

`get_oos_likelihood` *Out-of-sample Log-Likelihood (Proper Scoring Rule)*

Description

This function computes the out-of-sample log-likelihood (a strictly proper scoring rule) for each test event under a fitted REM or DEM.

Usage

```
get_oos_likelihood(
  object,
  verbose = FALSE,
  edgelist_test,
  edgelist_train = NULL,
  baseline_method = c("last", "trend", "mean", "beginning"),
  loess_span = 0.75
)
```

Arguments

object	A redeem object (either rem or dem).
verbose	Logical; if 'TRUE', prints verbose output. Defaults to FALSE.
edgelist_test	A matrix or data frame of test events (timing, from, to, type).
edgelist_train	A matrix or data frame of train events (timing, from, to, type). Defaults to 'NULL', in which case it retrieves the training events from the 'object' or the preprocessed data.
baseline_method	Character; how to compute the fixed log-baseline intensity used for out-of-sample scoring. One of: '"last"' (uses the last estimated baseline value), '"trend"' (extrapolates a LOESS trend), '"mean"', or '"beginning"'. Defaults to '"last"'.
loess_span	Numeric; LOESS span (0, 1] passed to predict_baseline_trend when baseline_method = "trend". Defaults to 0.75.

Value

A numeric vector of log-likelihoods for each test event.

See Also

rem_object and dem_object for details on prediction methods.

---

get_ranking *Get ranking for test events (Out-of-Sample Goodness-of-Fit)*

Description

Evaluates the out-of-sample predictive performance of a fitted model on a test event sequence using a ranking-based Goodness-of-Fit (GoF) procedure.

Usage

```
get_ranking(
  object,
  verbose = FALSE,
  k_max = 1000,
  edgelist_test,
  edgelist_train = NULL,
  ties.method = c("average", "first", "last", "random", "max", "min"),
  return_probabilities = FALSE,
  baseline_method = c("trend", "mean", "last", "beginning"),
  loess_span = 0.75
)
```

Arguments

`object`	A `redeem` object (either `rem` or `dem`).
`verbose`	Logical; if 'TRUE', prints verbose output. Defaults to FALSE.
`k_max`	Maximum number of ranked pairs to return. Defaults to 1000.
`edgelist_test`	A matrix of test events (timing, from, to, type).
`edgelist_train`	A matrix of train events (timing, from, to, type). Defaults to NULL.
`ties.method`	Character; the method to handle ties when ranking event intensities, passed directly to `rank`. Defaults to `"average"`. One of: `"average"` Assigns the average of the ranks of all tied elements to each. `"first"` Breaks ties by the order they appear in the data structure. `"last"` Breaks ties by the reverse order of their appearance. `"random"` Breaks ties randomly, ensuring no systematic bias. `"max"` Assigns the maximum of the ranks of the tied elements to all. `"min"` Assigns the minimum of the ranks of the tied elements to all.
`return_probabilities`	Logical; if TRUE, returns the predicted probabilities/scores instead of recall curves. Defaults to FALSE.
`baseline_method`	Character; how to compute the fixed log-baseline intensity used for out-of-sample scoring. Defaults to `"trend"`. One of: `"trend"` Fit a LOESS trend to the estimated piecewise-constant log-baseline (`est_time`) over training time and extrapolate it to the start of the test period. This mirrors the trend decomposition used in the application plot script and typically yields a better forecast than a fixed mean. `"mean"` Use the simple mean of `est_time`. `"last"` Use the last estimated log-baseline value (i.e.\ the value from the most recent training interval). `"beginning"` Set the baseline to 0.
`loess_span`	Numeric; LOESS span (0, 1] passed to `predict_baseline_trend` when `baseline_method = "trend"`. Defaults to 0.75.

Details

For each event observed in the test period (`edgelist_test`), the function:

1. Determines the set of all potential candidate dyads (the risk set) at that event's timestamp.
2. Computes the predicted event intensities (or probabilities) for all candidate dyads using the fitted model's parameters and the network history up to that moment.
3. Ranks all candidate dyads in descending order of their predicted intensities.
4. Determines the rank of the actually observed dyad.

A well-fitting model will consistently assign higher intensities to the dyads that actually interact, ranking them near the top.

The function summarizes the rankings across all test events to compute:

- **Mean Reciprocal Rank (MRR)**: The average of the reciprocal ranks of the true dyads.
- **Recall at K**: The proportion of test events where the true dyad is ranked within the top K candidate dyads.
- **Precision at K**: The proportion of top K recommendations that correspond to true events.

Value

A `ranking_redeem` data frame with columns:

`Cutpoint` Integer value from 0 to `k_max`.

`Recall` The proportion of test events where the true dyad is ranked at or within the cutpoint.

`Precision` The precision value at the cutpoint.

Additionally, the returned object has the following attributes:

`"mrr"` Mean Reciprocal Rank (MRR) of the true dyads.

`"mean_rank"` Mean rank of the true dyads (excluding ranks > `k_max`).

`"median_rank"` Median rank of the true dyads (excluding ranks > `k_max`).

`"hits_summary"` A data frame summarizing Recall, Precision, and F1 values at K = 1, 5, 10, and 50.

`get_residuals` *Get residuals for model diagnostics (Cox-Snell Residuals)*

Description

Computes Cox-Snell residuals for a fitted model to diagnose goodness-of-fit and calibration.

Usage

```
get_residuals(object, get_0_1 = TRUE, get_1_0 = TRUE, raw = FALSE)
```

Arguments

`object`	A `redeem` object (either `rem` or `dem`).
`get_0_1`	Logical; if 'TRUE', computes residuals for the formation (0 -> 1) process. Defaults to TRUE.
`get_1_0`	Logical; if 'TRUE', computes residuals for the dissolution (1 -> 0) process. Defaults to TRUE.
`raw`	Logical; if 'TRUE', returns the raw Cox-Snell residuals. Defaults to FALSE.

Details

Cox-Snell residuals are a standard diagnostic tool for continuous-time survival models and counting processes. Under the true model specification, the integrated cumulative intensity computed up to the exact time of an observed event is distributed as a standard exponential random variable, i.e., $\Lambda_{ij}(t_k) \sim Exp(1)$.

Consequently, if the model is correctly specified:

- The empirical survival function of these residuals should closely match the theoretical survival function of a standard exponential distribution, $S(r) = \exp(-r)$.
- Deviations between the empirical Kaplan-Meier curve of the residuals and the theoretical exponential curve signal model misspecification, unmodeled dyadic heterogeneity, or non-stationarity.

The function can compute residuals for both the formation/incidence ($0 \rightarrow 1$) process and the dissolution/duration ($1 \rightarrow 0$) process.

Value

If 'raw = TRUE', a list containing the raw residuals for the selected process(es). If 'raw = FALSE', a list of data frames containing the Kaplan-Meier coordinates ('time', 'surv') and the corresponding 'theoretical' standard exponential survival values.

References

Cox, D. R., & Snell, E. J. (1968). A general definition of residuals. Journal of the Royal Statistical Society: Series B (Methodological), 30(2), 248-265.

`plot_baseline` *Plot the Estimated Baseline Intensity*

Description

Draws a step-function plot of the estimated piecewise-constant baseline intensity against time. The function dispatches to class-specific methods for `dem`, `rem`, and `redeem_result` objects.

Usage

```
plot_baseline(x, ...)
```

Arguments

x	A dem, rem, or redeem_result object produced by dem or rem.
...	Additional arguments passed to graphics::plot.

Value

The original object x is returned invisibly. Called primarily for its side effect of producing a plot.

---

predict_baseline_trend

Predict the baseline intensity trend at one or more future time points

---

Description

Decomposes the estimated piecewise-constant log-baseline (est_time) into a smooth trend component (via LOESS) and a seasonal/residual component, following the same approach used in the application plot script. The fitted trend is then *extrapolated* to target_times using predict.loess() so that the baseline used for out-of-sample scoring reflects the long-run level of activity rather than any arbitrary fixed value.

Usage

```
predict_baseline_trend(model, target_times, loess_span = 0.75)
```

Arguments

model	A redeem_result object with non-null est_time and time_changepoints fields.
target_times	Numeric vector: the times at which to predict the trend (typically the unique timestamps of the test events).
loess_span	Numeric; the span argument passed to stats::loess. Larger values give a smoother (more conservative) trend extrapolation. Defaults to 0.75.

Details

The decomposition mirrors the plot code in the application:

1. Build a data frame of (time, est_time) using the changepoints stored in model$time_changepoints. The first interval [0, changepoint_1) is given time = 0; each subsequent interval gets the corresponding changepoint value.
2. Fit LOESS on the log-scale est_time values.
3. Predict at each target_times; predictions are clamped to the range of the observed est_time to avoid wild extrapolation.

Value

A numeric vector of predicted log-baseline (trend component) values, one per element of `target_times`. Falls back to `mean(est_time)` for each time point if there are fewer than 3 observations or if the LOESS fit fails.

`redeem_terms` *redeem Model Terms*

Description

The help pages of `rem` and `dem` describe the model formulation and estimation details. This page documents all statistics available to be used in the model formulas, characterizing the intensities of event formation and dissolution.

In the `redeem` framework, models like DEM (fitted via `dem`) and REM (fitted via `rem`) are specified using R formulas. The right-hand side of these formulas defines the structural statistics and covariates, where each term must be specified separately as an explicit function call (e.g., `~ inertia() + reciprocity(window = 10)`).

All terms support an optional `transformation` argument f. The available transformations are:

- `"identity"` (default): $f(x) = x$
- `"log"`: $f(x) = \log(x + 1)$
- `"recip"`: $f(x) = 1/(x + 1)$
- `"bin"`: $f(x) = I(x > 0)$
- `"sig"`: sigmoid-like saturation, $f(x) = x/(x + K)$

Throughout, $N_{i,j}(t)$ denotes the cumulative number of events from i to j up to (but not including) time t; $N^w_{i,j}(t)$ is the windowed analogue on $(t-w,\ t)$; $d^{\text{out}}_i(t) = |\{l : N_{i,l}(t) > 0\}|$ is the historical out-degree of i; and $c^{\text{out}}_i(t) = \sum_l N_{i,l}(t)$ is the total event count sent by i. The superscript act indicates that the quantity is computed on the currently active DEM network.

The implemented terms are grouped into five categories:

1. **Baseline and Nuisance Terms**: Intercept, time-varying baseline, and degree fixed effects.
2. **Endogenous Dyadic Terms**: Inertia, reciprocity, interaction duration, and participation shifts.
3. **Triadic Closure and Shared Partners**: Common partners and triangle statistics.
4. **Degree and Centrality Statistics**: Actor degree and event count statistics.
5. **Exogenous Covariates**: Dyadic and monadic covariate terms.

Arguments

`K`	Numeric; the evaluation point or scaling/saturation factor for the sufficient statistic (default is 1).
`transformation`	Character; specifies the transformation to apply to the statistic. One of: • `"identity"` (default): $f(x) = x$

- "log": $f(x) = \log(x + 1)$
- "recip": $f(x) = 1/(x + 1)$
- "bin": $f(x) = I(x > 0)$
- "sig": sigmoid-like saturation, $f(x) = x/(x + K)$

event_stream	Optional matrix or data frame; an alternative event stream to use for calculating the statistic. If NULL (default), the modeled stream is used.
window	Numeric; time window for calculating the statistic (default Inf, i.e., use full history).
type	Character; the specific variation of the statistic or triangle type (e.g., "OSP", "ISP", "OTP", "ITP", "out_sender", "sum").
mode	Character; the participation shift mode (e.g., "ABBA", "ABBY").
data	For dyadic_cov, a numeric matrix of dimensions $N \times N$, a scalar applied globally, or a named list of matrices for time-varying covariates. For monadic_cov, a numeric vector of length N or a named list of vectors for time-varying covariates.
fun	A function taking two arguments fun(v_i, v_j) to generate dyadic values.
change_points	Optional numeric vector; time points for time-varying covariates if data is a list.
changepoints	Numeric vector; time points where the baseline intensity is allowed to change.
labels	Character vector; optional labels for the resulting time intervals.
history	Character; "general" for cumulative history or "current" for currently active events.
count	Logical; if TRUE, uses count-based (weighted) versions of degree statistics (default FALSE).
...	Arguments passed to the underlying initialization function.

Value

A redeem_term object (a list containing structural information about the statistic) to be used inside model formulas.

Multi-Stream Event Covariates

Most endogenous terms support covariates calculated from multiple event streams. By providing an event_stream argument to a term (e.g., inertia(event_stream = other_events)), users can model one event process while accounting for the history of another. The package automatically handles the splintering and union of these timelines.

Baseline and Nuisance Terms

- Intercept(): Intercept: Constant log-intensity baseline. $s_{i,j}(t) = 1$. Also available as intercept().
- baseline(changepoints, labels): Baseline: Stepwise constant log-baseline with user-specified change points $c_1 < c_2 < \ldots < c_K$. $s_{i,j}(t) = \sum_{k=1}^{K} I(t \in [c_k, c_{k+1}))$. Coefficients are treated as nuisance parameters.
- degree / degrees: Degree Fixed Effects: Node-specific sender and receiver baselines α_i and γ_j estimated via Minorization-Maximization (MM). Contribution to linear predictor: $\alpha_i + \gamma_j$. Treated as nuisance parameters.

Endogenous Dyadic Terms

- `inertia(transformation, K, event_stream, window)`: Inertia: Cumulative count of past events from i to j. $s_{i,j}(t) = f(N_{i,j}(t))$; windowed: $f(N^w_{i,j}(t))$.
- `reciprocity(transformation, K, event_stream, window)`: Reciprocity: Cumulative count of past events from j to i. $s_{i,j}(t) = f(N_{j,i}(t))$ **(Directed only)**.
- `current_interaction(transformation, K, event_stream)`: Duration: Time elapsed since the currently active event (i, j) started. $s_{i,j}(t) = f(t - t_{\text{start},i,j})$ **(DEM only)**. Alias: `duration()`.
- **Participation Shifts** (for two consecutive events $(A \to B) \to (C \to D)$, each statistic is 1 if the specified pattern holds, 0 otherwise; **REM only, Directed only**):
  - `psABBA(event_stream)`: PS-ABBA: Receiver responds to sender. $s_{C,D}(t) = I(C = B,\, D = A)$.
  - `psABBY(event_stream)`: PS-ABBY: Receiver initiates to a new target. $s_{C,D}(t) = I(C = B,\, D \neq A)$.
  - `psABAY(event_stream)`: PS-ABAY: Sender initiates to a new target. $s_{C,D}(t) = I(C = A,\, D \neq B)$.
  - `psABXA(event_stream)`: PS-ABXA: Outsider targets original sender. $s_{C,D}(t) = I(C \neq A, C \neq B,\, D = A)$.
  - `psABXB(event_stream)`: PS-ABXB: Outsider targets original receiver. $s_{C,D}(t) = I(C \neq A, C \neq B,\, D = B)$.
  - `psABXY(event_stream)`: PS-ABXY: Entirely new dyad. $s_{C,D}(t) = I(C \neq A, C \neq B,\, D \neq A, D \neq B)$.
  - `ps(mode, event_stream)`: PS Shorthand: Dispatches to one of the six participation shift statistics above based on `mode` (one of `"ABBA"`, `"ABBY"`, `"ABAY"`, `"ABXA"`, `"ABXB"`, `"ABXY"`).

Triadic Closure and Shared Partners

- `general_common_partners( transformation, K, type, event_stream, window)`: Historical Common Partners: Number of nodes k sharing a historical directed path of the specified type with both i and j. $s_{i,j}(t) = f(|CP^{\text{type}}_{i,j}(t)|)$.
  - `"OSP"` (Outgoing Shared Partner): $N_{i,k}(t) > 0$ and $N_{j,k}(t) > 0$.
  - `"ISP"` (Incoming Shared Partner): $N_{k,i}(t) > 0$ and $N_{k,j}(t) > 0$.
  - `"OTP"` (Outgoing Two-Path): $N_{i,k}(t) > 0$ and $N_{k,j}(t) > 0$.
  - `"ITP"` (Incoming Two-Path): $N_{k,i}(t) > 0$ and $N_{j,k}(t) > 0$.

  Aliases: `general_common_partner()`, `general_common_partner_OSP()`, `general_common_partner_ISP()`, `general_common_partner_OTP()`, `general_common_partner_ITP()`.
- `current_common_partners( transformation, K, type, event_stream)`: Active Common Partners: As `general_common_partners` but restricted to currently active edges. $s_{i,j}(t) = f(|CP^{\text{type,act}}_{i,j}(t)|)$ **(DEM only)**. Aliases: `current_common_partner()`, `current_common_partner_OSP()`, `current_common_partner_ISP()`, `current_common_partner_OTP()`, `current_common_partner_ITP()`.
- `general_triangle(transformation, K, type, event_stream, window)`: Historical Triangles: Number of closed triads around (i, j) in the historical event network of the specified type. $s_{i,j}(t) = f(|\Delta^{\text{type}}_{i,j}(t)|)$ **(Directed only)**.

- `current_triangle(transformation, K, type, event_stream)`: Active Triangles: As `general_triangle` but restricted to currently active edges. **(DEM only, Directed only)**.
- `common_partner(history, type, ...)`: Common Partner Shorthand: Dispatches to `general_common_partners()` (`history="general"`) or `current_common_partners()` (`history="current"`).
- `triangle(history, type, ...)`: Triangle Shorthand: Dispatches to `general_triangle()` (`history="general"`) or `current_triangle()` (`history="current"`).

Degree and Centrality Statistics

- `general_degree_out_sender( transformation, K, event_stream, window)`: Sender Out-Degree: Historical out-degree of sender i. $s_{i,j}(t) = f(d_i^{\text{out}}(t))$ **(Directed only)**.
- `general_degree_out_receiver( transformation, K, event_stream, window)`: Receiver Out-Degree: Historical out-degree of receiver j. $s_{i,j}(t) = f(d_j^{\text{out}}(t))$ **(Directed only)**.
- `general_degree_in_sender( transformation, K, event_stream, window)`: Sender In-Degree: Historical in-degree of sender i. $s_{i,j}(t) = f(d_i^{\text{in}}(t))$ **(Directed only)**.
- `general_degree_in_receiver( transformation, K, event_stream, window)`: Receiver In-Degree: Historical in-degree of receiver j. $s_{i,j}(t) = f(d_j^{\text{in}}(t))$ **(Directed only)**.
- `general_degree_sum(transformation, K, event_stream, window)`: Degree Sum: Sum of historical degrees of both endpoints. $s_{i,j}(t) = f(d_i(t) + d_j(t))$ **(Undirected only)**.
- `general_degree_absdiff( transformation, K, event_stream, window)`: Degree Absolute Difference: Absolute difference in historical degrees. $s_{i,j}(t) = f(|d_i(t) - d_j(t)|)$ **(Undirected only)**.
- `general_count_out_sender( transformation, K, event_stream, window)`: Sender Out-Count: Total events sent by sender i. $s_{i,j}(t) = f(c_i^{\text{out}}(t))$ **(Directed only)**.
- `general_count_out_receiver( transformation, K, event_stream, window)`: Receiver Out-Count: Total events sent by receiver j. $s_{i,j}(t) = f(c_j^{\text{out}}(t))$ **(Directed only)**.
- `general_count_in_sender( transformation, K, event_stream, window)`: Sender In-Count: Total events received by sender i. $s_{i,j}(t) = f(c_i^{\text{in}}(t))$ **(Directed only)**.
- `general_count_in_receiver( transformation, K, event_stream, window)`: Receiver In-Count: Total events received by receiver j. $s_{i,j}(t) = f(c_j^{\text{in}}(t))$ **(Directed only)**.
- `general_count_sum(transformation, K, event_stream, window)`: Count Sum: Sum of total event counts of both endpoints. $s_{i,j}(t) = f(c_i(t) + c_j(t))$ **(Undirected only)**.
- `general_count_absdiff( transformation, K, event_stream, window)`: Count Absolute Difference: Absolute difference in total event counts. $s_{i,j}(t) = f(|c_i(t) - c_j(t)|)$ **(Undirected only)**.
- `current_degree_out_sender(transformation, K, event_stream)`: Active Sender Out-Degree: Out-degree of i in the active DEM network. $s_{i,j}(t) = f(d_i^{\text{out,act}}(t))$ **(DEM only, Directed only)**.
- `current_degree_out_receiver( transformation, K, event_stream)`: Active Receiver Out-Degree: Out-degree of j in active DEM network. $s_{i,j}(t) = f(d_j^{\text{out,act}}(t))$ **(DEM only, Directed only)**.
- `current_degree_in_sender(transformation, K, event_stream)`: Active Sender In-Degree: In-degree of i in the active DEM network. $s_{i,j}(t) = f(d_i^{\text{in,act}}(t))$ **(DEM only, Directed only)**.

- `current_degree_in_receiver( transformation, K, event_stream)`: Active Receiver In-Degree: In-degree of j in active DEM network. $s_{i,j}(t) = f(d_j^{\text{in,act}}(t))$ **(DEM only, Directed only)**.
- `current_degree_sum(transformation, K, event_stream)`: Active Degree Sum: Sum of active degrees of both endpoints. $s_{i,j}(t) = f(d_i^{\text{act}}(t) + d_j^{\text{act}}(t))$ **(DEM only, Undirected only)**.
- `current_degree_absdiff(transformation, K, event_stream)`: Active Degree Absolute Difference: Absolute difference in active degrees. $s_{i,j}(t) = f(|d_i^{\text{act}}(t) - d_j^{\text{act}}(t)|)$ **(DEM only, Undirected only)**.
- `current_count_out_sender(transformation, K, event_stream)`: Active Sender Out-Count: Total active events sent by i. $s_{i,j}(t) = f(c_i^{\text{out,act}}(t))$ **(DEM only, Directed only)**.
- `current_count_out_receiver( transformation, K, event_stream)`: Active Receiver Out-Count: Total active events sent by j. $s_{i,j}(t) = f(c_j^{\text{out,act}}(t))$ **(DEM only, Directed only)**.
- `current_count_in_sender(transformation, K, event_stream)`: Active Sender In-Count: Total active events received by i. $s_{i,j}(t) = f(c_i^{\text{in,act}}(t))$ **(DEM only, Directed only)**.
- `current_count_in_receiver( transformation, K, event_stream)`: Active Receiver In-Count: Total active events received by j. $s_{i,j}(t) = f(c_j^{\text{in,act}}(t))$ **(DEM only, Directed only)**.
- `current_count_sum(transformation, K, event_stream)`: Active Count Sum: Sum of active event counts of both endpoints. $s_{i,j}(t) = f(c_i^{\text{act}}(t) + c_j^{\text{act}}(t))$ **(DEM only, Undirected only)**.
- `current_count_absdiff(transformation, K, event_stream)`: Active Count Absolute Difference: Absolute difference in active event counts. $s_{i,j}(t) = f(|c_i^{\text{act}}(t) - c_j^{\text{act}}(t)|)$ **(DEM only, Undirected only)**.
- `degree( history, type, count, transformation, K, event_stream, window)`: Degree Shorthand: Dispatches to the appropriate degree statistic based on `history` (`"general"` or `"current"`) and `type` (`"out_sender"`, `"out_receiver"`, `"in_sender"`, `"in_receiver"`, `"sum"`, `"absdiff"`). Set `count = TRUE` for weighted (count-based) variants. Alias: `degrees()`.
- `count(history, type, transformation, K, event_stream, window)`: Count Shorthand: Equivalent to `degree(..., count = TRUE)`.

Exogenous Covariates

- `dyadic_cov(data, change_points)`: Dyadic Covariate: Time-constant or time-varying external dyadic covariate matrix X. $s_{i,j}(t) = X_{i,j}(t)$.
- `monadic_cov(data, fun, change_points)`: Monadic Covariate: External monadic covariate vector x converted to a dyadic matrix via user-supplied function g. $s_{i,j}(t) = g(x_i(t), x_j(t))$.

---

rem *Relational Event Model (REM) Estimation*

Description

Estimates a Relational Event Model (REM) for network data, focusing on the incidence of discrete events between pairs of actors. See `dem` for the full Durational Event Model, which extends the REM to handle interactions with non-negligible duration.

Usage

```
rem(
  events,
  training_start = 0,
  exogenous_end = NULL,
  formula = NULL,
  n_nodes = NULL,
  directed = FALSE,
  coef = NULL,
  semiparametric = FALSE,
  control = control.redeem()
)
```

Arguments

`events`	A matrix of events with columns `time`, `from`, `to`, and optionally `type` (1 for start, 3 for exogenous changes).
`training_start`	Numeric; the time point at which to start the estimation. Defaults to 0.
`exogenous_end`	Numeric; optional end time for exogenous baseline changes. Defaults to NULL.
`formula`	A one-sided `formula` specifying the sufficient statistics to include in the intensity function. The right-hand side must be composed of terms from `redeem_terms`. For example: `~ inertia() + reciprocity() + degree`. An intercept (`~ 1`) is the minimal specification. Defaults to NULL.
`n_nodes`	Integer; the total number of actors in the network. If `NULL` (default), it is automatically identified based on the actors in the `events` set.
`directed`	Logical; whether the interaction events are directed. Defaults to FALSE.
`coef`	Numeric vector; initial coefficients for the model. If provided, this must be a concatenated vector of: 1. Core coefficients: values for sufficient statistics in the formula. 2. Degree coefficients (if `degree` is in the formula): a vector of length `n_nodes` (undirected) or `2 * n_nodes` (directed, sender effects first then receiver effects). 3. Baseline coefficients (if temporal changepoints are present): a vector of length equal to the number of baseline intervals (equal to number of changepoints if an intercept/degree is present, or changepoints + 1 if neither is present). Defaults to NULL, in which case default starting values are automatically computed.
`semiparametric`	Logical; whether to use a semiparametric baseline. Defaults to FALSE. See the 'Semiparametric Baseline' section for details.
`control`	A list of control parameters from `control.redeem`. Defaults to `control.redeem()`.

Details

The REM can be viewed as the incidence sub-model of the full `dem`, corresponding to the formation process $\lambda^{0\to1}$. It uses a counting process approach to estimate the influence of various covariates on the timing and occurrence of events, assuming that events are instantaneous points in time.

Value

An object of class `rem_object` containing model estimates and log-likelihoods. See `rem_object` for details on the components of the returned object and S3 methods.

Model Formulation

The Relational Event Model characterizes the instantaneous rate at which actor pair (i, j) initiates an event. Under the log-linear specification, the event intensity at time t is:

$$\lambda_{i,j}(t \mid \mathscr{H}_t, \theta) = \exp\big(s_{i,j}(\mathscr{H}_t)^\top \alpha + \beta_i + \beta_j + f(t, \gamma)\big)$$

where:

- $s_{i,j}(\mathscr{H}_t)$ is a vector of sufficient statistics computed from the event history $\mathscr{H}_t$; see `redeem_terms` for available terms.
- α is the vector of covariate effects.
- β_i and β_j are optional actor-specific baselines (sender and receiver sociality), included via the bare symbol `degree` in the formula.
- $f(t, \gamma)$ is an optional piecewise-constant temporal baseline, included via `baseline(changepoints)` in the formula.

Semiparametric Baseline

When `semiparametric = TRUE`, the temporal baseline rate of event occurrence is left completely unspecified, and the model parameters are estimated via the Cox partial likelihood using the `survival` package. In this path:

- Each observed event time is treated as a failure time, and all non-occurring dyads at that time constitute the risk set.
- The exact waiting times between events are conditioned away, meaning that inference is based solely on the sequence of events and the relative dyadic intensities.
- This approach is equivalent to the *ordered* (or *conditional*) REM likelihood introduced by Butts (2008). It is highly robust to temporal fluctuations and baseline misspecification since no piecewise baseline or changepoints need to be specified.
- **Limitations**: This path does **not** support the specialized scalable estimation of sender/receiver popularity effects (`degree`) or piecewise-constant temporal baselines.

References

Fritz, C., Rastelli, R., Fop, M., & Caimo, A. (2026). Scalable Durational Event Models: Application to Physical and Digital Interactions. arXiv:2504.00049.

Butts, C. T. (2008). A Relational Event Framework for Social Action. Sociological Methodology, 38(1), 155-200.

Examples

```
# Simulate some relational event data
n <- 20
events <- matrix(c(
  1.2, 1, 5,
  3.1, 2, 8,
  4.5, 1, 3
), ncol = 3, byrow = TRUE)
colnames(events) <- c("time", "from", "to")

# Estimate a simple REM
fit <- rem(
  events = events,
  n_nodes = n,
  formula = ~1,
  control = control.redeem(it_max = 50)
)
summary(fit)
```

`rem.simulate` *Simulate a Relational Event Model (REM)*

Description

Simulate a Relational Event Model (REM)

Usage

```
rem.simulate(
  events = matrix(0, 0, 4),
  formula,
  coef = NULL,
  coef_degree = 0,
  n_events = 0,
  time = 0,
  max_events = 4e+05,
  n_nodes,
  verbose = FALSE,
  baseline = NULL,
  seed = 123,
  block = 1,
  directed = FALSE
)
```

Arguments

`events`	A matrix representing the initial events with columns `time`, `from`, `to`, and optionally `type` (1 for start, 3 for exogenous changes). Defaults to an empty 4-column matrix.

formula	A one-sided formula specifying the sufficient statistics to include in the intensity function. The right-hand side must be composed of terms from redeem_terms. For example: ~ inertia() + reciprocity() + degree. An intercept (~ 1) is the minimal specification.
coef	Numeric vector; coefficients for the model. Defaults to NULL.
coef_degree	Numeric; degree coefficient. Defaults to 0.
n_events	Integer; number of events to simulate. Defaults to 0.
time	Numeric; simulation time. Defaults to 0.
max_events	Integer; maximum number of events. Defaults to 400000.
n_nodes	Integer; the total number of actors in the network.
verbose	Logical; if TRUE, prints progress information. Defaults to FALSE.
baseline	Numeric vector; baseline intensity values for intervals defined by changepoints. Defaults to NULL.
seed	Integer; random seed. Defaults to 123.
block	An integer vector of length n_nodes indicating the block/group assignment for each node, or a single value applied to all nodes. Defaults to 1. If multiple blocks are assigned, within-block interactions are suppressed (i.e., their dyadic intensities are set to 0), meaning only events occurring between actors in different blocks are simulated.
directed	Logical; whether the interaction events are directed. Defaults to FALSE.

Details

The block parameter allows the user to specify a partition of the nodes into different groups (blocks). When the vector contains more than one unique block identifier:

- The simulation suppresses all within-block dyad intensities by setting them to 0.
- Consequently, only events between nodes belonging to different blocks are generated (between-block interactions).
- If all nodes belong to the same block (e.g., if a single value or NULL is passed), no block-level constraints are applied, and all dyads are simulated according to the specified model formula.

Value

A matrix of simulated events.

Note

Multi-stream event models are currently not supported in simulation.

Examples

```
# Simulate events from a REM model structure
n <- 10
f1 <- ~ 1 + inertia(transformation = "identity")
```

```
# Simulating events
evs <- rem.simulate(
  formula = f1,
  n_nodes = n,
  time = 1.0,
  coef = c(1.0, 0.5),
  seed = 42,
  max_events = 100
)
head(evs)
```

`rem_object` *The rem Object*

Description

An object of class `rem` returned by the `rem` function, representing a fitted Relational Event Model.

Value

A `rem` object is a list containing the following components:

- `call`: The matched call.
- `event_numbers`: A vector containing the number of events.
- `model`: The fitted underlying model.
- `events`: The preprocessed event matrix.
- `formula`: The formula used.
- `n_nodes`: The number of nodes.
- `directed`: Logical indicating whether the events are directed.
- `build_time`: The time at which the estimation dataset started building.
- `max_time`: The maximum event time.
- `time_changepoints`: Time points where baseline intensity changes.
- `labels_changepoints`: Labels for the time intervals.
- `training_start`: The start time of the training period.
- `exogenous_end`: The end time of the exogenous period.
- `subsample`: Subsample proportion used.
- `return_data`: Logical indicating whether preprocessed data frames were returned.
- `runtime`: The estimation runtime.
- `window_map`: The window map used for calculation.
- `preprocessed`: Preprocessed data structures.

Methods (S3)

The following S3 methods are implemented for `rem` objects:

- `print(x, ...)`: Prints a brief summary of the fitted REM model.
  - `x`: A `rem` object.
  - `...`: Additional arguments passed to printing function.
- `summary(object, ...)`: Summarizes model results, including parameter estimates, standard errors, and fit statistics. Returns an object of class `summary.rem`.
  - `object`: A `rem` object.
  - `...`: Additional arguments passed to summary method.
- `plot(x, baseline = FALSE, ...)`: Generates trace plots for model coefficients and log-likelihood histories.
  - `x`: A `rem` object.
  - `baseline`: Logical. If `TRUE`, plots the estimated baseline step function. Defaults to `FALSE`.
  - `...`: Additional arguments passed to underlying trace plotting method. Supported parameters include:
    - `coefs`: Logical. If `TRUE` (default), plots iteration traces for core/fixed coefficients.
    - `degree`: Logical. If `TRUE` (default), plots iteration traces for degree/actor effects.
    - `time`: Logical. If `TRUE` (default), plots iteration traces for temporal baseline effects.
    - `llh`: Logical. If `TRUE` (default), plots trace of log-likelihood history.
    - `separate`: Logical. If `TRUE`, each plot is generated in a new window or as a separate sequence. Defaults to `FALSE`.
    - `sub_label`: Character. Optional subtitle to display at the bottom of the figure.
- `logLik(object, ...)`: Extracts the log-likelihood of the fitted model.
  - `object`: A `rem` object.
  - `...`: Additional arguments.
- `plot_baseline(x, ...)`: Plots the step function of the estimated baseline intensity.
  - `x`: A `rem` object.
  - `...`: Additional graphical parameters passed to `plot`.
- `predict(object, time = NULL, type = c("response", "lp", "terms"), ...)`: Predicts the intensity, linear predictor, or term contributions for a fitted REM model.
  - `object`: A `rem` object.
  - `time`: Numeric vector; optional time point(s) at which to predict. Defaults to NULL.
  - `type`: Character; the type of prediction. Defaults to `"response"`.
  - `...`: Additional arguments.

summary.redeem_result *Summary of a* redeem_result *Model Fit*

Description

Computes a summary of a fitted redeem_result object, collecting the estimated fixed effects, log-likelihood, and (if present) degree and temporal baseline effects into a structured list suitable for printing.

Usage

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

Arguments

object	A 'redeem_result' object.
...	Additional arguments (currently unused).

Value

An object of class summary.redeem_result, which is a list containing:

- coefficients: A numeric matrix with one row per fixed-effect covariate and columns Estimate, Std. Error, t value, and Pr(>|t|).
- llh: The log-likelihood of the fitted model (scalar).
- degree_summary: A list with summary statistics (n, n_unidentifiable, mean, sd, range) of the estimated degree effects, only present when more than 10 degree parameters were estimated.
- degree_effects: A named numeric vector of estimated degree effects, only present when 10 or fewer degree parameters were estimated.
- time_summary: A list with summary statistics of the estimated temporal baseline effects, only present when more than 10 time intervals were used.
- time_effects: A named numeric vector of estimated temporal baseline effects, only present when 10 or fewer time intervals were used.
- iter: Integer; the number of iterations performed by the optimizer (NA if history was not saved).

Index