Package: remstimate (via r-universe)

October 28, 2026

Type Package

Title Optimization Frameworks for Tie-Oriented and Actor-Oriented Relational Event Models

Version 3.1.0

Date 2026-07-16

Maintainer Giuseppe Arena <g.arena@uva.nl>

Description Tools for fitting, diagnosing, and analyzing tie-oriented and actor-oriented relational event models, under both frequentist and Bayesian approaches. The package supports tie-oriented modeling (Butts, 2008, <doi:10.1111/j.1467-9531.2008.00203.x>) and an actor-oriented modeling framework (Stadtfeld et al., 2017, <doi:10.15195/v4.a14>), with additional model diagnostics and goodness-of-fit tools. Interfaces to estimation backends provide a range of extensions: random-effects (frailty) relational event models capturing sender, receiver, and dyadic heterogeneity (Juozaitiene & Wit 2024, <doi:10.1007/s11336-024-09952-x>; Mulder & Hoff, 2024, <doi:10.1214/24-AOAS1885>), finite mixture and dyadic latent class models for unobserved dyadic heterogeneity (Lakdawala et al., 2026, <doi:10.1016/j.socnet.2026.06.006>), penalized estimation via the lasso, ridge, and elastic net (Tibshirani, R., 1996, <doi:10.1111/j.2517-6161.1996.tb02080.x>; Karimova et al., 2023, <doi:10.1016/j.socnet.2023.02.006>), and approximate Bayesian regularization (Karimova et al., 2025, <doi:10.1016/j.jmp.2025.102925>). Modeling of events with a duration is also supported (Lakdawala et al., 2026, <doi:10.48550/arXiv.2602.21000>) and moving window relational event models (Mulder & Leenders, 2019, <doi:10.1016/j.chaos.2018.11.027>; Meijerink et al., 2023, <doi:10.1371/journal.pone.0272309>).

License MIT + file LICENSE

URL https://tilburgnetworkgroup.github.io/remstimate/

BugReports https://github.com/TilburgNetworkGroup/remstimate/issues

Depends R (>= 4.0.0), remify (>= 4.1.0), remstats (>= 4.1.0)

Imports Rcpp, trust, mvnfast

Suggests knitr, rmarkdown, tinytest, survival, coxme, glmnet, lme4, glmmTMB, flexmix, shrinkem (>= 0.4.0), MASS, nnet, remdata (>= 0.2.1)

LinkingTo Rcpp, RcppArmadillo, remify (>= 4.1.0)

VignetteBuilder knitr

Encoding UTF-8

LazyData true

RoxygenNote 7.3.3

LazyDataCompression gzip

NeedsCompilation yes

Author Giuseppe Arena [aut, cre] (ORCID: <https://orcid.org/0000-0001-5204-3326>), Joris Mulder [aut], Rumana Lakdawala [aut], Fabio Generoso Vieira [aut], Marlyne Meijerink-Bosman [ctb], Diana Karimova [ctb], Mahdi Shafiee Kamalabad [ctb], Roger Leenders [ctb]

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

Date/Publication 2026-07-17 16:20:02 UTC

RemoteUrl https://github.com/cran/remstimate

RemoteRef HEAD

RemoteSha 89d323459c3db3da812a823aa522a68886194b6f

Contents

AIC.remstimate . . . 3
AICC . . . 3
ao_data . . . 4
BIC.remstimate . . . 6
bic_table . . . 6
diagnostics . . . 7
dlcrem . . . 9
plot.diagnostics . . . 10
plot.remstimate . . . 11
print.remstimate . . . 12
remfrailty . . . 13
remixture . . . 15
rempenalty . . . 17
remstimate . . . 18
remtribute . . . 21
remwindow . . . 24
summary.remstimate . . . 26

tie_data 28
WAIC 29

Index **31**

`AIC.remstimate` *AIC for remstimate objects*

Description

Returns the AIC (Akaike's Information Criterion) value of a 'remstimate' object.

Usage

```
## S3 method for class 'remstimate'
AIC(object, ...)
```

Arguments

`object`	a `remstimate` object.
`...`	further arguments passed to `AIC`.

Value

AIC value.

`AICC` *AICC*

Description

A function that returns the AICC (Akaike's Information Corrected Criterion) value in a 'remstimate' object.

Usage

```
AICC(object, ...)

## S3 method for class 'remstimate'
AICC(object, ...)
```

Arguments

`object`	is a `remstimate` object.
`...`	further arguments to be passed to the 'AICC' method.

Value

AICC value of a 'remstimate' object.

Methods (by class)

- `AICC(remstimate)`: AICC (Akaike's Information Corrected Criterion) value of a 'remstimate' object

Examples

```
# ----------------------------------- #
#       tie-oriented model: "MLE"     #
# ----------------------------------- #

# loading data
data(tie_data)

# processing event sequence with remify
tie_reh <- remify::remify(edgelist = tie_data$edgelist, model = "tie")

# specifying linear predictor
tie_model <- ~ 1 +
               remstats::indegreeSender()+
               remstats::inertia()+
               remstats::reciprocity()

# calculating statistics
tie_reh_stats <- remstats::remstats(reh = tie_reh,
                                    tie_effects = tie_model)

# running estimation
tie_mle <- remstimate::remstimate(reh = tie_reh,
                                  stats = tie_reh_stats,
                                  method = "MLE",
                                  ncores = 1)

# AICC
AICC(tie_mle)
```

`ao_data` *Actor-Oriented Relational Event History*

Description

A randomly generated sequence of relational events with 5 actors and 100 events. The event sequence is generated by following an actor-oriented modeling approach (for more information on the algorithm used for the generation, refer to `help(topic = remulateActor, package = "remulate")` or `?remulate::remulateActor`).

Usage

```
data(ao_data)
```

Format

ao_data is a list object containing the following objects:

edgelist a data.frame with the raw simulated edgelist. The columns of the data.frame are:

- time the timestamp indicating the time at which each event occurred
- actor1 the actor that generated the relational event
- actor2 the actor that received the relational event

seed the seed value used in remulate::remulateActor() for generating the event sequence

true.pars a list of two vectors named "rate_model" and "choice_model", each containing the values of the parameters used in the generation of the event sequence

Examples

```
# (1) load the data into the workspace
data(ao_data)

# (2) process event sequence with \code{remify}
ao_reh <- remify::remify(edgelist = ao_data$edgelist, model = "actor")

# (3) define linear predictor and claculate stastistcs with \code{remstats} package

## linear predictor for the rate model
rate_model <- ~ 1 + remstats::indegreeSender()

## linear predictror for the choice model
choice_model <- ~ remstats::inertia() + remstats::reciprocity()

## calculate statistics
ao_reh_stats <- remstats::remstats(reh = ao_reh, sender_effects = rate_model,
receiver_effects = choice_model)

# (4) estimate model using method = "MLE" and print out summary

## estimate model
mle_ao <- remstimate::remstimate(reh = ao_reh, stats = ao_reh_stats, method = "MLE")

## print out a summary of the estimation
summary(mle_ao)
```

BIC.remstimate *BIC for remstimate objects*

Description

Returns the BIC (Bayesian Information Criterion) value of a 'remstimate' object.

Usage

```
## S3 method for class 'remstimate'
BIC(object, ...)
```

Arguments

object	a remstimate object.
...	further arguments passed to BIC.

Value

BIC value.

bic_table *Compare BIC across a list of MIXREM fits*

Description

Compare BIC across a list of MIXREM fits

Usage

```
bic_table(x, ...)
```

Arguments

x	A remstimate_mixrem_list returned when k is a vector in remstimate(..., method = "MIXREM").
...	Unused.

Value

A data frame with columns k, BIC, and delta_BIC, sorted by ascending BIC.

Examples

```
library(remstimate)
data(history, package = "remstats")
data(info,    package = "remstats")
colnames(history)[colnames(history) == "setting"] <- "type"

reh   <- remify::remify(edgelist = history[1:100, ], model = "tie",
                        riskset = "active")
stats <- remstats::tomstats(~ inertia(consider_type = FALSE), reh = reh,
                            attr_actors = info)

fits <- remstimate(reh, stats,
                   mixture = list(random = ~ (1 + inertia | dyad), k = 2:5))
bic_table(fits)
```

`diagnostics` *Compute the diagnostics of a* `remstimate` *object. Diagnostics are based on point estimates, also for Bayesian fit.*

Description

Compute the diagnostics of a `remstimate` object. Diagnostics are based on point estimates, also for Bayesian fit.

Usage

```
diagnostics(object, reh, stats, ...)

## S3 method for class 'remstimate'
diagnostics(object, reh, stats, top_pct = 0.05, surprise_threshold = 0.2, ...)
```

Arguments

`object`	is a `remstimate` object.
`reh`	is a `remify` object, the same used for the 'remstimate' object.
`stats`	is a `remstats` object, the same used for the 'remstimate' object.
`...`	further arguments to be passed to the 'diagnostics' method.
`top_pct`	numeric scalar in (0,1): threshold for the top-percentage recall summary (default 0.05).
`surprise_threshold`	numeric scalar in (0,1): rows of the recall table with `rel_rank` <= `surprise_threshold` are collected into `$surprises` (default 0.20). Lower `rel_rank` means the observed outcome was ranked further from the top of the predicted probabilities, i.e. more surprising.

Details

Every recall table (`$recall$per_event` for tie/actor, `$recall_joint`/`$recall_start`/`$recall_end` for durem) and every `$surprises` table derived from it shares this column layout:

event Row position within the analyzed subset (tie/actor only).

edgelist_id_row The corresponding row index into `reh$edgelist_id` (`event` resolved to the full, untrimmed object), computed as `start_stop[1] - 1 + event`.

sub_index Which simultaneous observation at that `event` this row is, when multiple events share a time point.

obs_id The observed outcome as an ID. Tie model: the observed dyad ID. Actor sender submodel: the observed sender actor ID. Actor receiver submodel: the observed receiver actor ID (see `sender_id` below for the paired sender).

sender_id, obs_label, sender_label, dyad_label Present on `$surprises` only. Human-readable resolutions of `obs_id` (and, for the actor receiver submodel, the paired sender), added for readability – not present on the raw `$recall` tables.

rel_rank `1 - rank/D_t`: 1 = observed outcome ranked most likely of all risk-set alternatives; 0 = ranked last (maximally surprising). `rank` is the average rank, so outcomes tied in predicted probability share the mean of their ranks; a model that cannot discriminate (all probabilities equal) scores `rel_rank` at chance (~0.5), rather than being inflated toward 1 by the arbitrary storage order of the risk set.

cum_prob Cumulative predicted probability of everything strictly more likely than the observed outcome, plus the observed outcome's own probability (ties are not double-counted).

obs_prob Predicted probability assigned to the observed outcome.

prob_ratio `obs_prob * D_t`: ratio vs. uniform/random guessing. Near 1 means the model had no real signal for this outcome (e.g. a cold-start actor/dyad); well below 1 means the model actively favored a different outcome (a genuine misspecification signal).

log_loss `-log(obs_prob)`, the per-observation negative log-likelihood contribution.

D_t Size of the risk set (number of eligible alternatives) at that decision. For durem end-process recall, epochs with `D_t = 1` are excluded by default, since a single-candidate softmax is always exactly 1 regardless of model fit and carries no diagnostic information.

`$surprise_offenders` (tie, actor sender/receiver) and `$surprise_offenders_dyad` (actor receiver only) tabulate, per dyad or actor, how often it appears among surprises relative to how often it was actually observed (`prop = n_surprises / n_total`), sorted worst first.

For durem, `obs_id`/`event`/`sub_index` are replaced by `eidx`, a row index directly into `reh$edgelist` (i.e. already resolved to the original input edgelist row – no offset arithmetic needed). `$surprise_offenders_joint`/`_start` tabulate, per dyad (`"actor1 -> actor2"`), how often it appears among surprises relative to how often it was actually observed in that recall table.

Methods (by class)

- `diagnostics(remstimate)`: diagnostics of a 'remstimate' object

`dlcrem` *Dyadic Latent Class REM*

Description

Fits a mixture REM where dyads are assigned to `k` latent classes, each with class-specific coefficients. Special case of `remixture` with a dyad-level random intercept (`~ (1 | dyad)`).

Usage

```
dlcrem(reh, stats, k = 2L, nrep = 3L, ...)
```

Arguments

`reh`	A `remify` object.
`stats`	A `remstats` object.
`k`	Number of latent classes (default 2).
`nrep`	Number of random restarts (default 3).
`...`	Additional arguments passed to `remstimate`.

Value

A `remstimate_mixrem` object.

References

Lakdawala, R., Leenders, R., & Mulder, J. (2026). Not all bonds are created equal: Dyadic latent class models for relational event data. *Social Networks*. doi:10.1016/j.socnet.2026.06.006

See Also

`remixture` for the general mixture interface.

Examples

```
data(tie_data)
reh <- remify::remify(edgelist = tie_data$edgelist, model = "tie")
stats <- remstats::remstats(reh = reh,
  tie_effects = ~ 1 + remstats::inertia() + remstats::reciprocity())
fit_dlcrem <- dlcrem(reh, stats)
```

plot.diagnostics *Plot diagnostics of a* remstimate *object*

Description

Produces diagnostic plots from an object returned by diagnostics(). Plot 1 (waiting-time Q-Q) and plot 2 (Schoenfeld residuals) are computed from the diagnostics object alone. Plot 3 (recall scatter) is also derived from the diagnostics object. Plots 4 and 5 (posterior density and trace plots) are HMC-only and additionally require the original remstimate fit via object.

Usage

```
## S3 method for class 'diagnostics'
plot(
  x,
  object = NULL,
  which = c(1:3),
  effects = NULL,
  sender_effects = NULL,
  receiver_effects = NULL,
  n_per_page = 4L,
  ...
)
```

Arguments

x	a diagnostics object returned by diagnostics().
object	optional remstimate fit. Required for plots 4–5 (HMC posterior diagnostics); ignored otherwise.
which	integer vector of plots to produce. Default 1:3 covers waiting times, Schoenfeld residuals, and recall. Add 4 or 5 for HMC posterior density and trace plots.
effects	character vector of effect names to include (tie model). NULL uses all available effects.
sender_effects	character vector of sender-model effects (actor model). Pass NA to skip the sender model entirely.
receiver_effects	character vector of receiver-model effects (actor model). Pass NA to skip the receiver model entirely.
n_per_page	integer; maximum panels per page for multi-effect plots (Schoenfeld, posterior density, trace). Default 4L (2x2 grid).
...	further graphical arguments (currently unused).

Value

x invisibly.

`plot.remstimate` *Plot method for* `remstimate` *objects*

Description

Backward-compatible wrapper that computes `diagnostics()` when needed and delegates all plotting to `plot.diagnostics`. Existing call signatures continue to work unchanged.

Usage

```
## S3 method for class 'remstimate'
plot(
  x,
  reh,
  diagnostics = NULL,
  which = c(1:5),
  effects = NULL,
  sender_effects = NULL,
  receiver_effects = NULL,
  ...
)
```

Arguments

`x`	a `remstimate` object.
`reh`	a `remify` object used for the estimation.
`diagnostics`	optional pre-computed diagnostics object of class `c("diagnostics","remstimate")`. If `NULL`, `stats` must be supplied via `....`
`which`	integer vector selecting plots (default `1:4`).
`effects`	character vector of effects to plot (tie model).
`sender_effects`	character vector of sender-model effects (actor model).
`receiver_effects`	character vector of receiver-model effects (actor model).
`...`	pass `stats` here when `diagnostics = NULL`.

Value

`x` invisibly.

`print.remstimate` *Print out a quick overview of a* `remstimate` *object*

Description

A function that prints out the estimates returned by a 'remstimate' object.

Usage

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

Arguments

`x`	is a `remstimate` object.
`...`	further arguments to be passed to the print method.

Value

no return value. Prints out the main characteristics of a 'remstimate' object.

Examples

```
# ----------------------------------- #
#       method 'print' for the        #
# ----------------------------------- #

# loading data
data(ao_data)

# processing event sequence with remify
ao_reh <- remify::remify(edgelist = ao_data$edgelist, model = "actor")

# specifying linear predictor (for sender rate and receiver choice model)
rate_model <- ~ 1 + remstats::indegreeSender()
choice_model <- ~ remstats::inertia() + remstats::reciprocity()

# calculating statistics
ao_reh_stats <- remstats::remstats(reh = ao_reh,
                                   sender_effects = rate_model,
                                   receiver_effects = choice_model)

# running estimation
ao_mle <- remstimate::remstimate(reh = ao_reh,
                                 stats = ao_reh_stats,
                                 method = "MLE",
                                 nsim = 100,
                                 ncores = 1)
```

```
# print
ao_mle

# ----------------------------------- #
#   for more examples check vignettes  #
# ----------------------------------- #
```

`remfrailty` *Frailty REM (actor / dyad random intercepts)*

Description

Convenience wrapper around `remstimate` that fits a GLMM with the default *frailty* structure: random intercepts capturing unobserved heterogeneity in actor activity and attractiveness. `remfrailty` only builds this default structure; for custom random-effects (random slopes, dyad-level intercepts, etc.) call `remstimate(..., random = ...)` directly.

Usage

```
remfrailty(
  reh,
  stats,
  approach = c("frequentist", "Bayesian"),
  engine = "auto",
  ...
)

frailty_rem(reh, stats, ...)
```

Arguments

`reh`	A `remify` or `remify_durem` object.
`stats`	A `remstats` object (`tomstats`, `aomstats`, or `remstats_durem`).
`approach`	Either `"frequentist"` or `"Bayesian"`. Bayesian frailty is not offered in this version.
`engine`	For `frequentist`, the interval-model backend: `"glmmTMB"` or `"lme4"`. The default `"auto"` uses `glmmTMB` when installed, otherwise `lme4`. Ignored for ordinal models, which always use `coxme`.
`...`	Additional arguments passed to `remstimate`.

Details

For **directed tie-oriented models**, each actor receives a random intercept for sending (`actor1`) and receiving (`actor2`), i.e. `(1 | actor1) + (1 | actor2)`, capturing sender activity and receiver attractiveness. Actor-level frailty is **not** identified for undirected tie models (`actor1`/`actor2` are an arbitrary dyad ordering, not sender/receiver), so for undirected data `remfrailty()` falls back

to symmetric dyad-level frailty (`1 | dyad`) and emits a message. (An explicit `remstimate(..., random = ~ (1 | actor1) + (1 | actor2))` on undirected data still errors.)

For **actor-oriented models**, the sender rate model receives `(1 | actor1)` and the receiver choice model receives `(1 | actor2)`.

For **duration models** (`remify_durem`), random intercepts are crossed with the `process` indicator (start vs end). When `extend_riskset_by_type = TRUE` (tie-oriented only), they are further crossed with the event `type`.

For interval timing, estimation uses `lme4` or `glmmTMB` (Poisson GLMM). For ordinal timing or actor-oriented receiver choice models, `coxme` is used automatically.

Value

A `remstimate_glmm` object. See `remstimate` for details on the return structure.

References

Juozaitiene, R., & Wit, E. C. (2024). Nodal heterogeneity can induce ghost triadic effects in relational event models. *Psychometrika*, 89(1), 151-171. doi:10.1007/s1133602409952x

Mulder, J., & Hoff, P. D. (2024). A latent variable approach for modeling relational data with multiple receivers. *Annals of Applied Statistics*. doi:10.1214/24AOAS1885

See Also

`remstimate` for the general estimation interface (and for custom random-effects structures via `random = ...`), `remixture` / `dlcrem` for mixture models.

Examples

```
# GLMM fits stack the full case-control design and call lme4, which is slow
# on realistic data, so the examples are shown (via \dontrun) not executed.
## Not run:
# Tie-oriented frailty
\donttest{
library(remstimate)

data(tie_data)
reh <- remify::remify(edgelist = tie_data$edgelist, model = "tie")
stats <- remstats::remstats(reh = reh,
  tie_effects = ~ 1 + remstats::inertia() + remstats::reciprocity())

# sender frailty
fit_s <- remstimate(reh, stats, random = ~ (1 | actor1))
fit_s

# sender + receiver frailty
fit_sr <- remstimate(reh, stats,
                     random = ~ (1 | actor1) + (1 | actor2))
fit_sr

# inspect random effects (accessor matches the engine used, glmmTMB by default)
```

```
glmmTMB::ranef(fit_sr$backend_fit)
glmmTMB::VarCorr(fit_sr$backend_fit)

# diagnostics and recall
# recall uses the random effects (BLUPs); Schoenfeld residuals and the
# waiting-time Q-Q are on the fixed effects, as for MLE.
diag_sr <- diagnostics(fit_sr, reh, stats)
diag_sr
plot(diag_sr, which = 1:3)

# plot 7: random-effects normality Q-Q
plot(fit_sr, reh, stats = stats, which = 7)

# glmmTMB engine (faster for large data, supports more complex structures)
fit_tmb <- remstimate(reh, stats,
                      random = ~ (1 | actor1), engine = "glmmTMB")
}

# Actor-oriented frailty
reh_ao <- remify::remify(tie_data$edgelist, model = "actor")
stats_ao <- remstats::remstats(reh_ao,
  sender_effects = ~ 1 + indegreeSender(),
  receiver_effects = ~ inertia())
fit_ao <- remfrailty(reh_ao, stats_ao)
summary(fit_ao)

## End(Not run)
```

`remixture` *Finite-mixture REM (latent classes)*

Description

Fits a finite-mixture relational event model: events are assigned to k latent classes, each with class-specific coefficients, via **flexmix**. This is the general mixture interface; `dlcrem` is the dyadic-latent-class special case.

Usage

```
remixture(reh, stats, random, k = 2L, concomitant = NULL, nrep = 3L, ...)
```

Arguments

`reh`	A `remify` object.
`stats`	A `remstats` object.
`random`	Clustering formula, e.g. `~ (1 | dyad)` or `~ (1 + inertia | dyad)`.
`k`	Number of latent classes (default 2).

concomitant	Optional concomitant formula for class probabilities.
nrep	Number of random restarts (default 3).
...	Additional arguments passed to remstimate.

Value

A remstimate_mixrem object.

References

Lakdawala, R., Leenders, R., & Mulder, J. (2026). Not all bonds are created equal: Dyadic latent class models for relational event data. *Social Networks*. doi:10.1016/j.socnet.2026.06.006

See Also

dlcrem, remstimate.

Examples

```
library(remstimate)
data(tie_data)
reh <- remify::remify(edgelist = tie_data$edgelist, model = "tie")
stats <- remstats::remstats(reh = reh,
  tie_effects = ~ 1 + remstats::inertia() + remstats::reciprocity())

# two-component mixture, dyads as clustering unit
fit2 <- remstimate(reh, stats,
                   mixture = list(random = ~ (1 + inertia | dyad), k = 2))
fit2

# coefficients: rows = effects, columns = components
fit2$coefficients
fit2$prior_probs

# which dyads ended up in which component?
flexmix::clusters(fit2$backend_fit)

# try k = 2:4 and compare via BIC
fits <- remstimate(reh, stats,
                   mixture = list(random = ~ (1 + inertia | dyad), k = 2:4))
bic_table(fits)
plot(fits)   # prints BIC table

# pick the best k and inspect
best <- fits[[ paste0("k", bic_table(fits)$k[1]) ]]
best

# diagnostics: per-component recall + combined
diag2 <- diagnostics(fit2, reh, stats)
diag2
plot(fit2, reh, stats = stats)
```

`rempenalty` *Penalized REM (lasso / ridge / elastic net; horseshoe when Bayesian)*

Description

Convenience wrapper around `remstimate` for regularised estimation. Frequentist fits use **glmnet**; Bayesian fits use **shrinkem** shrinkage priors. The baseline / intercept is never penalised (glmnet keeps it as its unpenalised intercept; shrinkem flags it via `unpenalized`).

Usage

```
rempenalty(
  reh,
  stats,
  approach = c("frequentist", "Bayesian"),
  alpha = 1,
  prior = "horseshoe",
  nfolds = 10L,
  lambda_select = c("1se", "min"),
  ...
)
```

Arguments

`reh`	A `remify` or `remify_durem` object.
`stats`	A `remstats` object.
`approach`	`"frequentist"` (glmnet) or `"Bayesian"` (shrinkem).
`alpha`	Elastic-net mixing for the frequentist fit: `1` = lasso (default), `0` = ridge.
`prior`	Shrinkage prior for the Bayesian fit (default `"horseshoe"`).
`nfolds`	Cross-validation folds (frequentist). Default `10`.
`lambda_select`	Which lambda (frequentist): `"1se"` (default) or `"min"`.
`...`	Additional arguments passed to `remstimate`.

Value

A `remstimate_glmnet` (frequentist) or `remstimate_shrinkem` (Bayesian) object.

References

Karimova, D., Leenders, R., Meijerink-Bosman, M., & Mulder, J. (2023). Separating the wheat from the chaff: Bayesian regularization in dynamic social networks. *Social Networks*, 74, 139-155. doi:10.1016/j.socnet.2023.02.006

Karimova, D., van Erp, S., Leenders, R., & Mulder, J. (2025). Honey, I shrunk the irrelevant effects! Simple and flexible approximate Bayesian regularization. *Journal of Mathematical Psychology*, 126, 102925. doi:10.1016/j.jmp.2025.102925

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1), 267-288. doi:10.1111/j.25176161.1996.tb02080.x

See Also

`remstimate`.

Examples

```
# ---- MLE for basic tie model ----
data(tie_data)
reh <- remify::remify(edgelist = tie_data$edgelist, model = "tie")
stats <- remstats::remstats(reh = reh,
  tie_effects = ~ 1 + remstats::inertia() + remstats::reciprocity())

# ---- Penalised (lasso) ----
fit_lasso <- remstimate(reh, stats, penalty = list(alpha = 1))

# ---- Penalised using Bayesian horseshoe (shrinkem) ----
fit_horseshoe <- remstimate(reh, stats, penalty = list(prior = "horseshoe"))
```

`remstimate` *remstimate*

Description

Fit a relational event model, using a `reh` object of class `remify` and a `stats` object of class `remstats`

The model structure is determined by the arguments:

- No structure arguments: basic REM (tie or actor, depending on `reh`/`stats`).
- `random`: mixed-effects model (GLMM).
- `penalty`: penalized relational event modeling (GLMNET or Bayesian via shrinkem).
- `mixture`: finite mixture model (MIXREM / flexmix).

The estimation approach is set via `approach`:

- `"frequentist"` (default): MLE, lme4/glmmTMB, glmnet, or flexmix depending on the structure.
- `"Bayesian"`: HMC via C++ (basic models) or shrinkem (penalized models).

Usage

```
remstimate(
  reh,
  stats,
  approach = c("frequentist", "Bayesian"),
  random = NULL,
  penalty = NULL,
  mixture = NULL,
  engine = "auto",
  bayes = list(),
  seed = NULL,
  ncores = 1L,
  WAIC = FALSE,
  method = NULL,
  ...
)
```

Arguments

reh	A `remify` or `remify_durem` object.
stats	A `tomstats`, `tomstats_sampled`, `aomstats`, or `remstats_durem` object.
approach	`"frequentist"` (default) or `"Bayesian"`.
random	One-sided formula for random effects, e.g. `~ (1 | actor1) + (1 | actor2)`. Fits a GLMM (lme4/glmmTMB, or coxme for ordinal models).
penalty	List of penalisation settings. Frequentist uses glmnet; Bayesian uses shrinkem. Recognised elements: `alpha` (elastic-net mixing, `1` = lasso (default), `0` = ridge), `nfolds` (CV folds, default `10`), `lambda_select` (`"1se"` (default) or `"min"`), `unpenalized` / `penalized` (see below), and `prior` (shrinkem prior, default `"horseshoe"`). By default the intercept / baseline structure is left unpenalised: any statistic that is an indicator (all values in {0,1}) is exempt, which covers the overall `baseline`, the duration start/end process intercepts (`baseline.start` / `baseline.end`) and fixed-effect type dummies (e.g. `FEtype_*`); count and continuous effect statistics are penalised. Adjust this default with two additive controls (both character vectors of statistic names): `unpenalized` *adds* names to the exemption, and `penalized` *removes* names from it, i.e. forces them back into the penalty even though they are 0/1 indicators (e.g. a p-shift dummy such as `psABAB.end` that is a genuine effect, not an intercept). `penalized` takes precedence when a name is given in both. Names must match the model statistics exactly (as printed by the remstats object, e.g. `"psABAB.end"`, not `"psABAB"`); a name that matches nothing is ignored with a warning.
mixture	List of finite-mixture settings (flexmix). Recognised elements: `k` (components, default `2`), `random` (clustering formula, e.g. `~ (1 | dyad)`), `concomitant` (optional concomitant formula), `nrep` (random restarts, default `3`). E.g. `list(k = 2, random = ~ (1 | dyad))` fits the dyadic latent class model (Lakdawala et al., 2026).

engine	GLMM backend: "glmmTMB" or "lme4"; ordinal models automatically use coxme. The default "auto" selects "glmmTMB" when installed (more robust on the stacked tie-oriented design), otherwise falls back to "lme4".
bayes	List of Bayesian (C++ HMC) controls for basic tie/actor models. Recognised elements: nsim (post-burnin iterations, default 2000), nchains (default 2), burnin (default 1000), thin (default 1), init (initial values, default MLE estimates), L (leapfrog steps, default 50), epsilon (leapfrog step size, default 0.002), prior (list with mean and vcov), and nsimWAIC (WAIC draws, default 100).
seed	Random seed.
ncores	Number of threads (C++ backends). Default 1L.
WAIC	Compute WAIC? Default FALSE.
method	[*Deprecated*] Legacy argument for backward compatibility. Use approach instead. Accepts "MLE" or "HMC" for basic models.
...	Further arguments passed to the backend. The former top-level knobs (alpha, nfolds, lambda_select, k, concomitant, nrep, nsim, nchains, burnin, thin, init, L, epsilon, prior, nsimWAIC) are *deprecated* but still accepted here and routed into penalty / mixture / bayes.

Value

A remstimate S3 object.

References

Butts, C. T. (2008). A relational event framework for social action. *Sociological Methodology*, 38(1), 155-200. doi:10.1111/j.14679531.2008.00203.x

Stadtfeld, C., & Block, P. (2017). Interactions, actors, and time: Dynamic network actor models for relational events. *Sociological Science*, 4, 318-352. doi:10.15195/v4.a14

Juozaitiene, R., & Wit, E. C. (2024). Nodal heterogeneity can induce ghost triadic effects in relational event models. *Psychometrika*, 89(1), 151-171. doi:10.1007/s1133602409952x

Mulder, J., & Hoff, P. D. (2024). A latent variable approach for modeling relational data with multiple receivers. *Annals of Applied Statistics*. doi:10.1214/24AOAS1885

Lakdawala, R., Leenders, R., & Mulder, J. (2026). Not all bonds are created equal: Dyadic latent class models for relational event data. *Social Networks*. doi:10.1016/j.socnet.2026.06.006

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1), 267-288. doi:10.1111/j.25176161.1996.tb02080.x

Karimova, D., Leenders, R., Meijerink-Bosman, M., & Mulder, J. (2023). Separating the wheat from the chaff: Bayesian regularization in dynamic social networks. *Social Networks*, 74, 139-155. doi:10.1016/j.socnet.2023.02.006

Karimova, D., van Erp, S., Leenders, R., & Mulder, J. (2025). Honey, I shrunk the irrelevant effects! Simple and flexible approximate Bayesian regularization. *Journal of Mathematical Psychology*, 126, 102925. doi:10.1016/j.jmp.2025.102925

Lakdawala, R., Leenders, R., Ejbye-Ernst, P., & Mulder, J. (2026). Modelling interaction duration in relational event models. *arXiv preprint*. doi:10.48550/arXiv.2602.21000

Examples

```
# ---- MLE for basic tie model ----
data(tie_data)
reh <- remify::remify(edgelist = tie_data$edgelist, model = "tie")
stats <- remstats::remstats(reh = reh,
  tie_effects = ~ 1 + remstats::inertia() + remstats::reciprocity())
fit <- remstimate(reh, stats)
summary(fit)


# ---- MLE for a model of events with a duration (tie-oriented only) ----
# (the baboons dataset is provided by the 'remdata' package)
if (requireNamespace("remdata", quietly = TRUE)) {
  data(baboons_obs, package = "remdata")
  reh_dur <- remify::remify(baboons_obs$edgelist[1:1000,], model = "tie",
    directed = FALSE, duration = TRUE)
  remstats_dur <- remstats::remstats(reh_dur,
    start_effects = ~ inertia(scaling = "std") +
      activeDegreeDyad(scaling = "std"),
    end_effects = ~ totaldegreeDyad(scaling = "std"),
    first = 50)
  remstimate_dur <- remstimate(reh_dur, remstats_dur)
  summary(remstimate_dur)
  diagnos_dur <- diagnostics(remstimate_dur, reh_dur, remstats_dur)
  plot(diagnos_dur)
}

# ---- Random effects (frequentist) ----
fit_glmm <- remstimate(reh, stats,
  random = ~ (1 | actor1) + (1 | actor2))

# ---- Penalised (lasso) ----
fit_lasso <- remstimate(reh, stats, penalty = list(alpha = 1))

# ---- Penalised using Bayesian horseshoe (shrinkem) ----
fit_horseshoe <- remstimate(reh, stats, penalty = list(prior = "horseshoe"))

# ---- Mixture ----
fit_mix <- remstimate(reh, stats,
  mixture = list(k = 2, random = ~ (1 + inertia | dyad)))
```

`remtribute` *remtribute*

Description

Model an event attribute (type, weight, or any event-level attribute) conditional on the observed dyad sequence. This implements the conditional decomposition of Brandes, Lerner & Snijders

(2009): first model which dyads interact (via remstimate), then model the event attribute given the observed dyad.

The reh object can be either a tie-oriented or an actor-oriented REM regardless of how the dyad sequence was modeled upstream. Because the event attribute is modeled given the observed dyad (which can be modeled using a tie model or actor model), the attribute model always uses *tie-oriented* (dyad-level) statistics as predictor variables.

Statistics can be supplied in two ways: either pass a pre-computed tomstats object via stats, or pass a one-sided formula via effects and let remtribute compute the statistics internally. When the reh was created with model = "actor", the effects route is recommended because tomstats requires a tie-oriented reh.

Usage

```
remtribute(
  reh,
  stats = NULL,
  effects = NULL,
  attribute = "type",
  attribute_type = c("nominal", "ordinal", "numeric"),
  attr_actors = NULL,
  memory = "full",
  memory_value = Inf,
  ...
)
```

Arguments

reh	A remify object whose $edgelist contains the attribute column (carried through via event_type or event_attributes in remify). Accepted for both model = "tie" and model = "actor".
stats	A tomstats object (M x D x p), or NULL. When provided, statistics are subsetted internally to the observed dyad at each event. Ignored when effects is supplied.
effects	A one-sided formula with tie-model effects, e.g. ~ inertia() + reciprocity(). When provided, tomstats is called internally to compute the statistics. This is the recommended interface when reh uses an actor-oriented model, because it avoids the need for a separate tie-oriented reh.
attribute	Character string: column name in reh$edgelist that contains the event attribute to model. Default "type".
attribute_type	The type of the attribute: "nominal" (unordered categories, multinomial logit), "ordinal" (ordered categories, cumulative link model), or "numeric" (continuous, linear regression).
attr_actors	Optional data frame of actor-level covariates, passed to tomstats when effects is used.
memory	Memory type for statistics computation. Passed to tomstats when effects is used. Default "full".
memory_value	Memory parameter value. Passed to tomstats when effects is used.

... Additional arguments passed to the fitting backend (`nnet::multinom`, `MASS::polr`, or `stats::glm`).

Details

The function extracts the event attribute from the remified edgelist and, for each event, subsets the statistics array to the observed dyad. This yields an M x p design matrix of dyad-level covariates. The attribute is then modeled as a function of these covariates using the appropriate backend:

- `nominal`: `nnet::multinom`
- `ordinal`: `MASS::polr`
- `numeric`: `stats::glm` (Gaussian)

Note that even when the upstream event model is actor-oriented (sender rate + receiver choice), the attribute model uses `tomstats` (tie-oriented) statistics. This is because `aomstats` decomposes statistics into sender-level and receiver-level arrays, while the attribute model conditions on the full observed dyad and requires dyad-level covariates.

Value

An object of class `remtribute` containing the fitted model, coefficients, and metadata.

References

Brandes, U., Lerner, J., & Snijders, T. A. B. (2009). Networks evolving step by step: Statistical analysis of dyadic event data. In *2009 International Conference on Advances in Social Network Analysis and Mining* (pp. 200–205). IEEE.

Arena, G., Mulder, J., & Leenders, R. Th. A. J. (2023). Weighting the past: An extended relational event model for negative and positive events. *Journal of the Royal Statistical Society Series A*, 189(1), 359–381.

Examples

```
# --- Example 1: Tie model + pre-computed stats ---
if (requireNamespace("remdata", quietly = TRUE)) {
  data(hypertext, package = "remdata")
  # numeric attribute
  reh_text <- remify::remify(hypertext[1:1000,], model = "tie",
    directed = FALSE, riskset = "active",
    event_attributes = "duration")
  remstats_text <- remstats::remstats(reh_text,
    tie_effects = ~ inertia(scaling = "std") + totaldegreeDyad(scaling = "std"),
    first = 50)
  # fit model for event rate
  remstimate_tie <- remstimate(reh_text, remstats_text)
  # fit model for event attribute
  fit_attr <- remtribute(reh_text, stats = remstats_text, attribute = "duration",
   attribute_type = "numeric")

  # nominal attribute
  hypertext$long <- hypertext$duration>20
```

```
  reh_text <- remify::remify(hypertext[1:1000,], model = "tie", directed = FALSE,
    riskset = "active", event_attributes = "long")
  remstats_text <- remstats::remstats(reh_text,
    tie_effects = ~ inertia(scaling = "std") +
    totaldegreeDyad(scaling = "std"),
    first = 50)
  remstimate_text <- remstimate(reh_text, remstats_text)
  summary(remstimate_text)
  fit_attr <- remtribute(reh_text, stats = remstats_text, attribute = "long",
    attribute_type = "nominal")

  # --- Example 2: Actor model + effects formula ---
  reh_act <- remify::remify(edgelist = hypertext[1:1000,], model = "actor",
                            event_attributes = "duration")
  stats_act <- remstats::remstats(reh_act, sender_effects = ~ outdegreeSender(),
    receiver_effects = ~ inertia())
  fit_act <- remstimate(reh_act, stats_act)

  # Use effects formula -- no need for a separate tie-model reh
  fit_type <- remtribute(reh_act, effects = ~ inertia() + reciprocity(),
    attribute = "duration",
    attribute_type = "numeric")
  summary(fit_type)
}
```

---

remwindow *Moving-window REM estimation*

Description

Fits a relational event model repeatedly over slices of events, reusing a single `remify` object and full `remstats` object throughout. Each window's design is a row-slice of the stats array/list carrying an updated `subset` attribute; `remstimate` uses that attribute to pull the matching interevent times / dyad or actor IDs from the (unsliced) `reh`.

Usage

```
remwindow(
  reh,
  stats,
  n.windows = 5L,
  window.width = NULL,
  step.size.window = NULL,
  start.point = 1L,
  min.events = 50L,
  approach = c("frequentist", "Bayesian"),
  parallel = FALSE,
  ncores.window = 1L,
```

```
  ...
)
```

Arguments

reh	A remify object.
stats	A tomstats, tomstats_sampled, or aomstats object built on the full reh.
n.windows	Target number of windows in auto mode. Default 5.
window.width	Fixed number of events per window (manual mode).
step.size.window	Events to advance between windows (manual mode only). Default equals window.width.
start.point	First array row to start windowing from. Default 1L.
min.events	Floor on window width (auto and manual).
approach	"frequentist" (default) or "Bayesian", forwarded to remstimate.
parallel	Fit windows in parallel? Default FALSE.
ncores.window	Parallel workers across windows (Unix/macOS only; falls back to sequential with a message on Windows).
...	Further arguments passed to remstimate for every window.

Details

In **auto mode** (window.width = NULL, the default), n.windows equal contiguous slices spanning the full event range are used, reduced automatically (with a message) if that would put any window below the required minimum width. In **manual mode** (window.width supplied), windows have a fixed size and advance by step.size.window (default equal to window.width, i.e. non-overlapping); the final window absorbs any remainder so no events are dropped.

Value

A remstimate_window object: list(fits, windows, call, type, mode, n.windows).

References

Meijerink-Bosman, M., Leenders, R., & Mulder, J. (2022). Dynamic relational event modeling: Testing, exploring, and applying. PLoS One, 17(8). doi:10.1371/journal.pone.0272309

Mulder, J., & Leenders, R. T. A. (2019). Modeling the evolution of interaction behavior in social networks: A dynamic relational event approach for real-time analysis. Chaos, Solitons & Fractals, 119, 73-85. doi:10.1016/j.chaos.2018.11.027

Examples

```
# ---- MLE for basic tie model ----
data(tie_data)
reh <- remify::remify(edgelist = tie_data$edgelist, model = "tie")
stats <- remstats::remstats(reh = reh,
  tie_effects = ~ 1 + remstats::inertia() + remstats::reciprocity())
fit <- remstimate(reh, stats)
```

```
summary(fit)


# ---- MLE for a model of events with a duration (tie-oriented only) ----
# (the baboons dataset is provided by the 'remdata' package)
if (requireNamespace("remdata", quietly = TRUE)) {
  data(baboons_obs, package = "remdata")
  reh_dur <- remify::remify(baboons_obs$edgelist[1:1000,], model = "tie",
    directed = FALSE, duration = TRUE)
  remstats_dur <- remstats::remstats(reh_dur,
    start_effects = ~ inertia(scaling = "std") +
      activeDegreeDyad(scaling = "std"),
    end_effects = ~ totaldegreeDyad(scaling = "std"),
    first = 50)
  remstimate_dur <- remstimate(reh_dur, remstats_dur)
  summary(remstimate_dur)
  diagnos_dur <- diagnostics(remstimate_dur, reh_dur, remstats_dur)
  plot(diagnos_dur)
}

# ---- Random effects (frequentist) ----
fit_glmm <- remstimate(reh, stats,
  random = ~ (1 | actor1) + (1 | actor2))

# ---- Penalised (lasso) ----
fit_lasso <- remstimate(reh, stats, penalty = list(alpha = 1))

# ---- Penalised using Bayesian horseshoe (shrinkem) ----
fit_horseshoe <- remstimate(reh, stats, penalty = list(prior = "horseshoe"))

# ---- Mixture ----
fit_mix <- remstimate(reh, stats,
  mixture = list(k = 2, random = ~ (1 + inertia | dyad)))
```

`summary.remstimate` *Generate the summary of a* `remstimate` *object*

Description

A function that returns the summary of a `remstimate` object.

Usage

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

Arguments

`object`	is a `remstimate` object.
`...`	further arguments to be passed to the 'summary' method.

Value

no return value. Prints out the summary of a 'remstimate' object. The output can be save in a list, which contains the information printed out by the summary method.

Examples

```
# ----------------------------------- #
#       method 'summary' for the      #
#         tie-oriented model.        #
# ----------------------------------- #

# loading data
data(tie_data)

# processing event sequence with remify
tie_reh <- remify::remify(edgelist = tie_data$edgelist, model = "tie")

# specifying linear predictor
tie_model <- ~ 1 +
               remstats::indegreeSender()+
               remstats::inertia()+
               remstats::reciprocity()

# calculating statistics
tie_reh_stats <- remstats::remstats(reh = tie_reh,
                                    tie_effects = tie_model)

# running estimation
tie_mle <- remstimate::remstimate(reh = tie_reh,
                                  stats = tie_reh_stats,
                                  method = "MLE",
                                  nsim = 100,
                                  ncores = 1)

# summary
summary(tie_mle)

# ----------------------------------- #
#      method 'summary' for the       #
#      actor-oriented model: "MLE"    #
# ----------------------------------- #

# loading data
data(ao_data)

# processing event sequence with remify
```

```
ao_reh <- remify::remify(edgelist = ao_data$edgelist, model = "actor")

# specifying linear predictor (for sender rate and receiver choice model)
rate_model <- ~ 1 + remstats::indegreeSender()
choice_model <- ~ remstats::inertia() + remstats::reciprocity()

# calculating statistics
ao_reh_stats <- remstats::remstats(reh = ao_reh,
                                   sender_effects = rate_model,
                                   receiver_effects = choice_model)

# running estimation
ao_mle <- remstimate::remstimate(reh = ao_reh,
                                 stats = ao_reh_stats,
                                 method = "MLE",
                                 nsim = 100,
                                 ncores = 1)

# summary
summary(ao_mle)

# ----------------------------------- #
#   for more examples check vignettes  #
# ----------------------------------- #
```

`tie_data` *Tie-Oriented Relational Event History*

Description

A randomly generated sequence of relational events with 5 actors and 100 events. The event sequence is generated by following a tie-oriented modeling approach (for more information run on console `help(topic = remulateTie, package = "remulate")` or `?remulate::remulateTie`).

Usage

```
data(tie_data)
```

Format

`tie_data` is a list object containing the following objects:

`edgelist` a `data.frame` with the raw simulated edgelist. The columns of the `data.frame` are:

- `time` the timestamp indicating the time at which each event occurred
- `actor1` the actor that generated the relational event
- `actor2` the actor that received the relational event

`seed` the seed value used in `remulate::remulateTie()` for generating the event sequence

`true.pars` a vector containing the values of the parameters used in the generation of the event sequence

Examples

```
# (1) load the data into the workspace
data(tie_data)

# (2) process event sequence with \code{remify}
tie_reh <- remify::remify(edgelist = tie_data$edgelist, model = "tie")

# (3) define linear predictor and claculate stastistcs with \code{remstats} package

## linear predictor
tie_model <- ~ 1 + remstats::indegreeSender() + remstats::inertia() + remstats::reciprocity()

## calculate statistics
tie_reh_stats <- remstats::remstats(reh = tie_reh, tie_effects = tie_model)

# (4) estimate model using method = "MLE" and print out summary

## estimate model
mle_tie <- remstimate::remstimate(reh = tie_reh, stats = tie_reh_stats, method = "MLE")

## print out a summary of the estimation
summary(mle_tie)
```

`WAIC` *WAIC*

Description

A function that returns the WAIC (Watanabe-Akaike's Information Criterion) value in a 'remstimate' object.

Usage

```
WAIC(object, ...)

## S3 method for class 'remstimate'
WAIC(object, ...)
```

Arguments

`object`	is a `remstimate` object.
`...`	further arguments to be passed to the 'WAIC' method.

Value

WAIC value of a 'remstimate' object.

Methods (by class)

- `WAIC(remstimate)`: WAIC (Watanabe-Akaike's Information Criterion) value of a 'remstimate' object

Examples

```
# No examples available at the moment
```

Index

∗ **datasets**
ao_data, 4
tie_data, 28

AIC, *3*
AIC.remstimate, 3
AICC, 3
ao_data, 4

BIC, *6*
BIC.remstimate, 6
bic_table, 6

diagnostics, 7
dlcrem, 9, *14–16*

frailty_rem *(*remfrailty*)*, 13

plot.diagnostics, 10
plot.remstimate, 11
print.remstimate, 12

remfrailty, 13
remify, *22*
remixture, *9*, *14*, 15
rempenalty, 17
remstimate, *9*, *13*, *14*, *16*, *17*, 18, *18*, *22*
remtribute, 21
remwindow, 24

summary.remstimate, 26

tie_data, 28

WAIC, 29